НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 044.77

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Спеціалізовані комп'ютерні системи

на тему: Евристичний метод багатокритеріальної оптимізації

Виконав (-ла): студент (-ка) ІІ курсу, групи КВ-73мп

Місік Дмитро Сергійович

Науковий керівник доцент кафедри, ктн, Зорін Ю.М.

Рецензент _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія

Спеціалізовані комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)

«___»_____2018р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Місіку Дмитру Сергійовичу

1. Тема дисертації: евристичний метод багатокритеріальної оптимізації, науковий керівник дисертації к.т.н., доцент Зорін Юрій Михайлович, затверджені наказом по університету від «30» жовтня 2018 р. №4030-с

2. Термін подання студентом дисертації 07 грудня 2018 р.

3. Об'єкт дослідження: процес багатокритеріальної оптимізації.

4. Предмет дослідження: евристичний метод розв'язання задачі багатокритеріальної оптимізації.

5. Перелік завдань, які потрібно розробити
   - опис предметної області досліджень та обґрунтування багатокритеріальної оптимізації;
   - евристичні алгоритми багатокритеріальної оптимізації;
   - модифікований алгоритм квіткового запилення для розв'язання задачі оптимізації.

6. Перелік ілюстративного матеріалу: 34 рисунка, 13 таблиць.

7. Перелік публікацій

– «Модифікований алгоритм квіткового запилення для розв'язання задачі глобальної оптимізації», X конференція молодих вчених ПМК-2018-1. – 2018;

– «Евристичний метод багатокритеріальної оптимізації», XI конференція молодих вчених ПМК-2018-2. – 2018;

8. Дата видачі завдання 5 вересня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вивчення літератури за тематикою проекту	05.09.2017	
2	Аналіз існуючих рішень	20.01.2018	
3	Підготовка матеріалів першого розділу магістерської дисертації	09.03.2018	
4	Підготовка матеріалів другого розділу магістерської дисертації	30.04.2018	
5	Підготовка матеріалів третього розділу магістерської дисертації	10.09.2018	
6	Підготовка графічної частини дипломного проекту	16.10.2018	
7	Оформлення документації дипломного проекту	01.11.2018	
8	Попередній розгляд магістерської дисертації на кафедрі	26.11.2018	

Студент _____ Місік Д.С.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Зорін Ю.М.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. З розвитком сучасних технологій стає необхідним швидкий та ефективний розв’язок складних математичних задач. Особливо популярними є задачі оптимізації. Проблема розв’язання оптимізаційних задач постає не лише в надсучасних комп’ютерних технологіях, а і в повсякденному людському житті. Дуже часто ці задачі зводяться до оптимізації певного об’єкту за групою критеріїв. Наприклад, в ряді задач економічної галузі необхідно максимально збільшити прибуток, але при цьому мінімізувати витрати. В задачі про комівояжера необхідно врахувати не лише відстань від початкового до кінцевого пункту, а і затримки, витрати, загальний час переїзду та певні інші критерії. Всі ці задачі можуть бути сформульовані в загальному випадку наступним чином: максимізувати/мінімізувати декілька функцій, що визначенні на певному проміжку розв’язків з певним критерієм оптимальності. Такий ряд задач називається багатокритеріальною задачею оптимального рівняння. Вона займає важливе місце серед задач теорії вибору та прийняття рішень. Кожен параметр оптимальності певного розв’язку може конфліктувати з іншим параметром оптимальності. Через це на сьогоднішній день існує багато підходів до розв’язання такого ряду задач та аналізу оптимальності знайденого розв’язку. Тому створення алгоритму, який може аналізувати групу параметрів та знаходити розв’язок, що задовольняє кожному з критеріїв є складною задачею і на сьогоднішній день.

Об’єкт дослідження є процес багатокритеріальної оптимізації.

Предметом дослідження є евристичні методи розв’язання задачі багатокритеріальної оптимізації.

Мета роботи є розробка нового методу для розв’язання задачі багатокритеріальної оптимізації, що характеризується вищою швидкодією та більшою точністю розв’язків, ніж відомі методи.

Методи дослідження. В роботі використовуються методи евристичних алгоритмів, методи дискретної математики, методи дослідження операцій, методи комбінаторної оптимізації.

Наукова новизна:

- Проаналізовано існуючі алгоритми оптимізації в неперервному просторі, показано їх недоліки в порівнянні з алгоритмом квіткового запилення;
- Запропоновано модифікацію існуючого алгоритму квіткового запилення для розв’язання задачі багатокритеріальної оптимізації;
- Виконано порівняльний аналіз оригінального алгоритму та модифікованого алгоритму квіткового запилення для розв’язання задачі багатокритеріальної оптимізації.

Практична цінність отриманих в роботі результатів полягає в тому, що розроблений модифікований алгоритм дозволяє прискорити процес багатокритеріальної оптимізації та покращити результат в сенсі точності і сталості результату. Крім того, алгоритм показав що здатний розв’язувати задачу багатокритеріальної оптимізації без будь-якої додаткової інформації, крім тої, що необхідна для обчислення цільової функції. Лише за допомогою методу скаляризації декількох функцій його було приведено до форми, що дозволяє розв’язувати задачу багатокритеріальної оптимізації.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п’яти розділів та висновків.

У *вступі* подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність роботи, сформульовану мету і задачу дослідження, показано наукову новизну і практичну цінність виконаної роботи.

У *першому розділі* розглянуто існуючі алгоритми розв’язання задачі оптимізації в неперервному просторі, їхні особливості, переваги та недоліки, їхні реалізації.

У *другому розділі* розглянуто алгоритм квіткового запилення для розв’язання задачі оптимізації в неперервному просторі та запропоновано модифікацію існуючого алгоритму для його покращення і розв’язання задачі багатокритеріальної оптимізації.

У *третьому розділі* наведено особливості реалізації розробленої системи.

У *четвертому розділі* проведено аналіз отриманих результатів та порівняння їх з результатами оригінального алгоритму.

У *п’ятому* розділі проведено результат науково-дослідної практики.

У *висновках* представленні результати проведеної роботи.

Робота представлена на 124 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: задача багатокритеріальної оптимізації, алгоритм квіткового запилення, ефективність за Парето, скаляризація.

ABSTRACT

Actuality of theme. With the development of modern technologies, it is necessary to quickly and efficiently solve complex mathematical problems. A lot of popularity receives optimization tasks. The problem of solving optimization tasks arises not only in the up-to-date computer technologies, but also in everyday human life. Very often these tasks are reduced to the optimization of an object by a group of criteria. For example, in several tasks in the economic sphere, it is necessary to maximize profit, but at the same time minimize costs. In the task of salesman should consider not only the distance from the start to the final point, but also the delays, costs, total travel time and certain other criteria. All these tasks can be formulated in the general case next: maximize/minimize several functions that are defined at a certain interval of decisions with a certain criterion of optimality. Such a set of problems is called the multicriteria problem of an optimal equation. It occupies an important place among the tasks of the theory of choice and decision-making. Each optimality parameter of a solution may conflict with another optimality parameter. Because of this, today there are many approaches to solving such problems and analyzing the optimality of the solution found. Therefore, creating a solution that can analyze a group of parameters and find a solution that satisfies each of the criteria is a complex task to date.

The object of the study is the process of multicriteria optimization.

The subject of the study is heuristic methods for solving the multicriteria optimization problem.

The purpose of the work is to develop a new method for solving the multicriterial optimization problem characterized by higher speed and greater accuracy of solutions than known methods.

Research methods. Methods of heuristic algorithms, methods of discrete mathematics, methods of investigation of operations, methods of combinatorial optimization are used in this work.

Scientific novelty:

- The existing optimization algorithms in the continuous space are analyzed, their disadvantages are compared with the algorithm of flower pollination;
- The modification of the existing algorithm of flower pollination for solving the optimization problem in a continuous space is proposed;
- A comparative analysis of the original algorithm and the modified algorithm of flower pollination for the solution of the multicriteria optimization problem has been performed.

The practical value of the results obtained in the work consists in the fact that the developed modified algorithm allows to improve the process of multi-criteria optimization and improve the result in terms of accuracy and sustainability of the result. In addition, the algorithm has shown that it can solve the multicriterial optimization problem without any additional information other than what is necessary for the calculation of the target function. Only using the scalarization method applied for several functions it was brought to a form that allows solving the problem of multi-criteria optimization.

Structure and scope of work. The master's thesis consists of an introduction, five chapters and conclusions.

The *introduction* gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the work, formulates the purpose and task of the study, shows the scientific novelty and practical value of the work performed.

In the *first section*, existing algorithms for solving the problem of optimization in a continuous space, their features, advantages and disadvantages, and their implementation are considered.

In the *second section* we consider the algorithm of flower pollination to solve the problem of optimization in a continuous space and proposed modification of the existing algorithm to improve it.

The *third section* presents the peculiarities of the implementation of the developed system.

In the *fourth section,* the analysis of the obtained results and comparison with the results of the original algorithm are carried out.

In *fifth section,* the analysis of scientific research practice.

The *conclusions* are presented by the results of the work.

The work is presented on 124 sheets of paper, contains a link to the list of used literary sources.

Keywords: problem of multicriteria optimization, algorithm of flower pollination, efficiency by Pareto, scalarization.

РЕФЕРАТ

Актуальность темы. С развитием современных технологий становится необходимым быстрое и эффективное решение сложных математических задач. Особенно популярны задачи оптимизации. Проблема решения оптимизационных задач возникает не только в сверхсовременных компьютерных технологиях, но и в повседневной человеческой жизни. Очень часто эти задачи сводятся к оптимизации определенного объекта с группой критериев. Например, в ряде задач экономической области необходимо максимально увеличить прибыль, но при этом минимизировать расходы. В задачи о коммивояжере необходимо учесть не только расстояние от начального до конечного пункта, а и задержки, расходы, общее время переезда и некоторые другие критерии. Все эти задачи могут быть сформулированы в общем случае следующим образом: максимизировать/минимизировать несколько функций, определенные на некотором промежутке решений по нескольким критериям оптимальности. Такой ряд задач называется многокритериальные задачи оптимального уравнения. Эта задача занимает важное место среди задач теории выбора и принятия решений. Каждый параметр оптимальности определенного решения может конфликтовать с другим параметром оптимальности. Поэтому на сегодняшний день существует много подходов к решению такого ряда задач и анализа оптимальности найденного решения. Поэтому создание решения, которое может анализировать группу параметров и находить решение, удовлетворяющее каждому из критериев, является сложной задачей и по сегодняшний день.

Объект исследования является процесс многокритериальной оптимизации.

Предметом исследования является эвристические методы решения задачи многокритериальной оптимизации.

Цель работы является разработка нового метода для решения задачи многокритериальной оптимизации, которая характеризуется более высоким быстродействием и большей точностью решений, чем известные методы.

Методы исследования. В работе используются методы эвристических алгоритмов, методы дискретной математики, методы исследования операций, методы комбинаторной оптимизации.

Научная новизна:

- Проанализированы существующие алгоритмы оптимизации в непрерывном пространстве, показано их недостатки по сравнению с алгоритмом цветочного опыления;
- Предложена модификация существующего алгоритма цветочного опыления для решения задачи оптимизации в непрерывном пространстве;
- Выполнен сравнительный анализ оригинального алгоритма и модифицированного алгоритма цветочного опыления для решения задачи многокритериальной оптимизации.

Практическая ценность полученных в работе результатов заключается в том, что разработан модифицированный алгоритм позволяет ускорить процесс многокритериальной оптимизации и улучшить результат в смысле точности и устойчивости результата. Кроме того, алгоритм показал, что способен решать задачу многокритериальной оптимизации без какой-либо дополнительной информации, кроме той, которая необходима для вычисления целевой функции. Только с помощью метода скаляризации нескольких функций был приведен к форме, что позволяет решать задачу многокритериальной оптимизации.

Структура и объем работы. Магистерская диссертация состоит из введения, пяти глав и выводов.

Во *введении* представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована

актуальность работы, сформулирована цель и задача исследования, показано научную новизну и практическую ценность выполненной работы.

В *первой главе* рассмотрены существующие алгоритмы решения задачи оптимизации в непрерывном пространстве, их особенности, преимущества и недостатки, их реализации.

Во *второй главе* рассмотрен алгоритм цветочного опыления для решения задачи оптимизации в непрерывном пространстве и предложены модификации существующего алгоритма для его улучшения.

В *третьей главе* приведены особенности реализации разработанной системы.

В *четвертой главе* проведен анализ полученных результатов и сравнение их с результатами оригинального алгоритма.

В *пятой главе* проведен анализ научно-исследовательской практики.

В *выводах* представлены результаты проведенной работы.

Работа представлена на 124 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: задача многокритериальной оптимизации, алгоритм цветочного опыления, эффективность по Парето, скаляризация.

ЗМІСТ

ДОДАТКИ

ДОДАТКИ

Додаток 1. Копія презентації

Додаток 2. Копія публікацій

Додаток 3. Фрагменти програмного коду

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

FPA (flower pollination algorithm) – алгоритм квіткового запилення.

MFPA (modified flower pollination algorithm) – модифікований алгоритм квіткового запилення.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

ОС – операційна система.

LINQ (Language Integrated Query) – бібліотека, що є частиною .NET Framework.

CIL (Common Intermediate Language) – загальна проміжна мова.

CLI (Common Language Infrastructure) – специфікація міжмовної інфраструктури.

CLR (Common Language Runtime) – вихідний файл виконання мов.

DLL (Dynamic-link library) – динамічно-приєднувана бібліотека

EXE (executable) – розширення виконуваного файлу.

WPF (Windows Presentation Foundation) – технологія розробки десктопних додатків від Microsoft.

WCF (Windows Communication Foundation) – протокол обміну даними від Microsoft.

ECMA (European Computer Manufacturers Association) – некомерційна асоціація європейських виробників комп’ютерів.

ISO (International Organization for Standardization) – міжнародна організація зі стандартизації.

IL (Intermediate Language) – проміжна мова.

NaN (Not-a-Number) – один зі станів числа з плаваючою комою.

CSV (Comma Separated Values) – файловий формат.

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

XML (Extensible Markup Language) – розширення мови розмітки.

HTML (Hypertext Markup Language) – мова розмітки.

ВСТУП

Багатокритеріальна оптимізація (або багатокритеріальне програмування) – це процес оптимізації декількох цільових функцій в заданій області визначення.

Задача багатокритеріальної оптимізації формується наступним чином:

$$mir\big(f_1(\bar{x}), f_2(\bar{x}), \dots, f_k(\bar{x})\big), \bar{x} \in S$$

$$f_i: R^n \to R, k \geq 2$$

Вектори розв'язків $\bar{x} = (x_1, x_2, \dots, x_n)^T$ є непустим розв'язком області визначення множини S

Задачею багатокритеріальної оптимізації є пошук такого вектору $\bar{x}$, що оптимізує всі функції $f_i(x)$.

Проблемою багатокритеріальної оптимізації є те, що один вектор розв'язку, що є оптимальним для певного набору функцій може бути не оптимальним для іншого набору функцій. Тому для цього існують певні критерії оптимальності знайденого розв'язку, що включають в аналіз всі функції оптимізації.

Алгоритм квіткового запилення (Flower pollination algorithm) є однією з найсучасніших метаевристик, яка використовується для розв'язання задачі глобальної оптимізації в неперервному просторі.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМІЗАЦІЇ ТА ОБГРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

1.1. Постановка задачі

Задачею багатокритеріальної оптимізації є пошук такого вектору рішень, за якого всі функції оптимізації приймають своє мінімальне значення. Але існують такі набори функцій оптимізації, що не мають оптимального розв'язку, який мінімізує одночасно всі критерії.

Отже для знаходження оптимального розв'язку необхідно дослідити критерії оптимальності знайденого розв'язку багатокритеріальної оптимізації. Цей критерій дозволить обрати розв'язок серед інших альтернативних критеріїв.

Математично задачу багатокритеріальної оптимізації можна записати наступним чином:

$$f_1(\bar{x}) \to min, f_2(\bar{x}) \to min, \dots, f_n(\bar{x}) \to min \quad (1)$$

$$\bar{x} = (x_1, x_2, \dots, x_k) \quad (2)$$

Область допустимих значень задається наступним чином:

$$\sum_{i=1}^{k} a_{ji} x_i \le b_j \, , j = 1..n \quad (3)$$

$$x_i \ge 0, j = 1..n \quad (4)$$

Функції оптимізації (1) також мають назву критерії оптимізації. В випадку якщо необхідно знайти максимум критеріїв оптимізації, тоді критерії f_i заміняються критеріями q_i такими, що

$$q_i = -f_i, i = 1..n \quad (5)$$

В такому випадку задача оптимізації виконується над функціями q_i.

Проблема оптимізації, а саме неоднозначність при виборі оптимального розв'язку, може бути описана наступною задачею. Нехай критеріями оптимізації є три наступні функції:

$$f_1 = x, f_2 = -x + C, f_3 = C/2 \quad (6)$$

$$0 \leq x \leq C \quad (7)$$

$$C > 0 \quad (8)$$

Перша функція приймає своє мінімальне значення в точці $x = 0$, друга в точці $x = C$, третя функція є постійною на всій області визначення. Тобто не існує такого єдиного значення аргументу, за якого всі критерії набувають оптимального значення.

Це означає що розв'язком багатокритеріальної задачі оптимізації є не мінімум або максимум одного з критеріїв, а їх рівновага.

1.2. Оптимальність за Парето

Один з критеріїв оптимальності знайденого вектору розв'язків є критерій Парето. Ефективність за Парето – це такий стан системи, при котрому будь-яке покращення одного з критеріїв не може бути виконано без погіршення інших показників. За словами Парето: «будь-яка зміна, яка не погіршує жодний з показників та поліпшує хоча б один показник є покращенням» [1].

Математично ефективність за Парето сформульовано наступним чином: вектор розв'язків $\bar{x} \in S$ називається ефективним за Парето, якщо не існує такого $\bar{x`} \in S$, при якому $f_i(\bar{x`}) \leq f_i(\bar{x})$ для кожного $i = 1,2, \dots k$ і при цьому $f_i(\bar{x`}) > f_i(\bar{x})$ для хоча б одного i, де k – кількість функцій оптимізації. Множина ефективних розв'язків за Парето позначається $P(S)$ – також називається Парето-фронтом.

Множина станів системи, які є оптимальними за Парето, називаються «множиною Парето». Іншою назвою цих станів є «компромісні» стани.

Критерій Парето часто застосовується в економічній сфері. Це є одним з центральних понять сучасної економіки. В цій сфері ефективність за Парето визначена трьома правилами:

1. Критерієм оптимальності є не сумарна максимізація всіх критеріїв, а максимальна користь для кожного суб'єкту з його наявними ресурсами;
2. Необхідною умовою досягнення оптимальності є забезпечення рівноваги, тобто досягнення такого стану системи за якого відхилення будь-якого з показників викликає погіршення користі хоча б одного з суб'єктів;
3. Покращення одного з критеріїв, що не викликає погіршення хоча б одного іншого критерія, є покращенням за Парето.

На перший погляд здається що ці правила можуть бути застосовані лише до економічної сфери, але це не є так. Ефективність за Парето може бути застосована і до багатокритеріальної оптимізації.

Для множини Парето справедливе наступне: кожен з критеріїв оптимізації є рівнозначним іншим критеріям.

1.3. Класичні методи розв'язання

1.3.1. Метод послідовної оптимізації

Один з найпростіших методів розв'язання багатокритеріальної задачі оптимізації є метод послідовної оптимізації.

Суть методу наступна: спочатку всі критерії оптимізації сортуються за значимістю. Після сортування набору функцій оптимізується перший критерій оптимальності, використовуючи метод однокритеріальної оптимізації. Після оптимізації першого критерія, встановлюється величина Δx_1, що характеризує максимальне відхилення цього критерія, та застосовується задача однокритеріальної оптимізації до другого критерія. Цей процес продовжується до моменту коли всі критерії оптимізовані.

Процес є може бути описаний наступним чином:

1. Відсортувати критерії оптимізації $f_i(x)$ таким чином, щоб $f_1(x)$ був найзначніший критерій, а $f_n(x)$ – найменше значимий;
2. Оптимізувати перший критерій, застосовуючи метод однокритеріальної оптимізації;
3. Встановити величину Δx_1, що характеризує максимальне абсолютне відхилення першого критерія;
4. Повторити кроки 2 і 3 для наступних критеріїв.

Головним недоліком методу є те, що в методі не визначено випадки, коли величину абсолютного відхилення необхідно застосовувати та в залежності від реалізації цього методу розв'язання, розв'язки можуть бути різними.

Також недоліком є те, що метод взагалі не має формування «значимості критеріїв», що також в залежності від реалізації може вплинути на результат розв'язання задачі.

1.3.2. Метод скаляризації

Один з методів розв'язання задачі багатокритеріальної оптимізації є метод розв'язання задачі багатокритеріальної оптимізації використовуючи загальний (інтегральний) критерій.

Метод описується наступним чином: декілька функцій оптимізації зводяться до однієї функції оптимізації, яка і є об'єктом оптимізації. Таким чином задача багатокритеріальної оптимізації зводиться до задачі однокритеріальної оптимізації. Для того щоб об'єднати декілька функцій оптимізації в одну використовують метод скаляризації.

Скаляризація – це метод розв'язання задачі багатокритеріальної оптимізації, коли множинна показників (критеріїв) зводиться до одного показника за допомогою функції скаляризації. Функція скаляризації – це цільова функція, що характеризує багатокритеріальну оптимізації [2].

Один з методів скаляризації є методом зваженої суми. Метод зваженої суми - це метод, що об’єднує всі функції оптимізації в одну функцію наступним чином:

$$F_1 = \sum_{i=1}^{K} w_i f_i \tag{1}$$

$$\sum_{i=1}^{K} w_i = 1 \tag{2}$$

де k – кількість функцій оптимізації, w_i – випадкова вага. Цей метод є адитивним тому що критерії приведені до одного за допомогою операції додавання.

Іншим методом скаляризації є метод зваженого добутку. Аналогічно до попереднього методу метод об’єднує функції за допомогою операції добутку наступним чином:

$$F_2 = \prod_{i=1}^{K} [f_i]^{w_i} \tag{3}$$

$$\sum_{i=1}^{K} w_i = 1 \tag{4}$$

де k – кількість функцій оптимізації, w_i – випадкова вага. Цей метод має і іншу назву – мультиплікативна скаляризація.

Більш складним методом скаляризації є канонічна адитивна-мультиплікативна скаляризація. Цей метод об’єднує два попередні методи в один метод. Канонічна адитивна-мультиплікативна скаляризація визначена наступним чином:

$$F_3 = \beta \sum_{i=1}^{K} w_i f_i + (1 - \beta) \prod_{i=1}^{K} [f_i]^{w_i} \tag{5}$$

$$\sum_{i=1}^{K} w_i = 1 \tag{6}$$

$$0 \leq \beta \leq 1 \tag{7}$$

де β – адаптаційний параметр, k – кількість функцій оптимізації, w_i – випадкова вага. В випадку коли $\beta = 0$ метод зводиться до методу взваженого добутку, а якщо $\beta = 1$, то формула аналогічна до методу взважених сум.

Ще один відомий метод скаляризації є показникова скаляризація. Вона виражається наступною формулою:

$$F_4 = \sum_{i=1}^{K} (1 - e^{-w_i f_i}) \tag{8}$$

$$\sum_{i=1}^{K} w_i = 1 \tag{9}$$

$$w_i = 1 - w_i \tag{10}$$

де k – кількість функцій оптимізації, w_i – випадкова вага.

1.3.3. Метод Парето

Наступним методом розв'язку задачі оптимізації є метод Парето. Повернемося до задачі багатокритеріальної оптимізації:

$$f_1(\bar{x}) \to min, f_2(\bar{x}) \to min, \dots, f_n(\bar{x}) \to min \tag{1}$$

$$\bar{x} = (x_1, x_2, \dots, x_k) \tag{2}$$

$$\sum_{i=1}^{k} a_{ji} x_i \leq b_j, j = 1..n \tag{3}$$

$$x_i \geq 0, j = 1..n \tag{4}$$

Парето запропонував наступний метод розв'язання задачі багатокритеріальної оптимізації. Спочатку для кожного з критеріїв f_i розв'яжемо наступну задачу однокритеріальної оптимізації:

$$\begin{cases} f_i \to min, i = 1..n \\ \sum_{i=1}^{k} a_{ji} x_i \le b_j, j = 1..n \\ x_i \ge 0, j = 1..n \end{cases} \quad (5)$$

В результаті отримаємо вектор $\overline{f^*}$ оптимальних значень для кожного з критеріїв. Далі розв'яжемо наступну задачу оптимізації:

$$\begin{cases} F = \sum_{i=1}^{n} \frac{a_i f_i}{f_i^*} \to min \\ \sum_{i=1}^{k} a_{ji} x_i \le b_j, j = 1..n \\ x_i \ge 0, j = 1..n \end{cases} \quad (6)$$

де a_i – випадкова вага функції f_i, причому така, що

$$\sum_{i=1}^{n} a_i = 1 \quad (7)$$

Оптимальний розв'язок останньої задачі однокритеріальної оптимізації має назву ефективна точка за Парето. Множина таких точок є множиною Парето. Розв'язок цієї задачі і є розв'язком задачі багатокритеріальної оптимізації [3].

Недоліками цього методу є те, що якщо дано n критеріїв оптимізації, то однокритеріальна оптимізація відбувається $n + 1$ разів, бо n разів оптимізуються початкові критерії, які потім зводяться до одного критерія, котрий також необхідно оптимізувати. Останнє приведення є адитивним методом скаляризації, де вага кожного з критеріїв $\frac{a_i}{f_i^*}$.

1.4. Застосування задачі багатокритеріальної оптимізації

Задача багатокритеріальної оптимізації в реальному житті постає у вигляді: «знайти найкращий об’єкт за певним набором критеріїв». Головною задачею будь-якого підприємства та бізнесу є заробити як найбільше грошей.

На підприємстві майбутній прибуток залежить від багатьох параметрів: кількість робочої сили, автоматизація підприємства, витрати на сировину, витрати на доставку, витрати на електроенергію, витрати на воду, собівартість товару, кількість виробленого товару і так далі. Серед цих показників є показники, які залежать один від одного, наприклад, від кількості робочої сили на підприємстві залежить кількість виробленого товару, від степені автоматизації підприємства залежить кількість робочої сили та витрати на електроенергію, від витрат на електроенергію залежить кінцевий прибуток підприємства. За всіх цих показників треба обрати такі, за якого прибуток підприємства буде максимальним.

При збільшенні степені автоматизації підприємства, буде зростати також і витрати на електроенергію та зменшуватись кількість робочої сили. Але машини не можна залишати без нагляду, бо техніка може вийти з ладу і лагодити її необхідно людині. Вже в цих критеріях є конфлікт. Отже для максимізації прибутку треба вирішити задачу багатокритеріальної оптимізації де треба зменшити кількість робочої сили, збільшити степінь автоматизації, зменшити витрати на воду і електроенергію і так далі.

Наступним прикладом багатокритеріальної задачі оптимізації є задача комівояжера. Задачею комівояжера є пройти через певну кількість міст, обравши оптимальний маршрут. Оптимальний маршрут – це той, що є найкоротшим і найдешевшим. Отже в задачі комівояжера є два критерії, що можуть суперечити один одному – довжина маршруту та його вартість. Зазвичай найкоротший маршрут не є найдешевшим і навпаки – найдешевший маршрут є не найкоротшим.

В повсякденному житті ми також розв’язуємо багатокритеріальні задачі оптимізації, наприклад, обрати продукт певної категорії, що є найкращим серед конкурентів та найдешевшим. Люди зазвичай купують продукцію відомих брендів, що мають найнижчу ціну. Але з математичної точки зору, треба обрати таку продукцію, що повинна відповідати щонайменше двом критеріям оптимальності: бути кращою серед конкурентів та найдешевшою.

Бути найкращою продукцією серед конкурентів насправді є комплексним критерієм, бо, наприклад, серед ноутбуків найкращий той, що має найпотужніший процесор, найпотужнішу відеокарту, найбільше пам’яті і так далі. Серед автомобілів найкращим є той, що витрачає найменше палива, має найбільшу швидкість і так далі. Насправді показник «найкращий» є суб’єктивним та кожен інтерпретує його по-різному. Тому така повсякденна і на перший погляд проста задача є задачею багатокритеріальної оптимізації з якнайменше двома критеріями.

1.5. Висновок

Серед вищенаведених методів розв’язання задачі багатокритеріальної оптимізації кожен з методів має свої недоліки та переваги. Метод простої скаляризації є простим у застосуванні та для виконання багатокритеріальної оптимізації необхідно лише привести критерії до єдиної функції та виконати однокритеріальну оптимізацію один раз. Це є його головною перевагою перед методом Парето. Метод послідовної оптимізації має дуже багато недоліків та має нечітке формулювання.

1.6. Метаеврестичні алгоритми оптимізації

На сьогоднішній день існує безліч метаевристичних методів оптимізації функції:

1.6.1. Метод рою часток

Це є один з методів численної оптимізації функції, для котрого не треба знати градієнта об’єкту оптимізації. Метод рою часток було доведено декількома вченими, серед яких Кеннеді, Ші та Еберхартон. Саме в книзі Кеннеді та Ебархарта описуються так званий «інтелект рою» та певні філософські аспекти зграї.

Вже більш детальне дослідження виконав Полі та Ші. Метод рою часток оптимізує функцію через те, що підтримує «життя» деякої кількості груп часток. Наступним етапом їх життя є переміщення в просторі, що підпорядковується певній математичній формулі. Переміщення відбувається за умови якщо воно було виконано в кращу позицію.

Даний алгоритм спочатку призначався для імітації соціальної поведінки рою, але у майбутньому було помічено, що його можна також і в цілях оптимізації функції.

Поведінку алгоритму добре характеризує поведінка будь-якої зграї, наприклад, пташок. Переміщуючись великими групами вони ніколи не стикаються один з одним, зграя рухається не швидко, скоординовано. В зграї птиць немає вожака, так само як і в колонії деяких комах, наприклад, зграя бджіл. Зграя діє лише за ройовим інтелектом. На прикладі птиць, кожна птаха слідкує за своєю ділянкою на землі і знайшовши щось цікаве, сповіщає всю зграю. Саме так зграя птахів швидко знаходить годівницю [4].

Той факт, що птахи спілкуються використовуючи «ройовий інтелект» було досліджено багатьма вченими. Найбільш популярне пояснення цієї поведінки є те, що рухаючись у великій зграї один птах не може помічати кожну деталь на землі. В випадку з їжею, вона лише випадковим чином розкидана по землі. З цього випливають і недоліки такого підходу: необхідність боротьби за знайдену їжу.

В 1986 році вище Крейг Рейнольдс надихнувся такою поведінкою зграї птахів та створив комп’ютерну модель, що була призначена для

імітації поведінки зграї птахів. Алгоритм отримав назву Boids. Алгоритм мав три головних правила: кожна птиця намагається уникнути зіткнення з іншим птахом, кожен птах рухається в тому ж напрямку, що і сусідній птах, птахи намагалися рухатися на однаковій відстані один від одного. Не дивлячись на простоту кожного з правил, моделювання показало, що птахи рухались так само як і в реальній зграї. Крейг відмітив, що розроблена їм модель може бути модифікована додатковими факторами, такими як їжа.

Вже в 1995 році вищезгадані вчені Джеймс Кеннеді та Рассел Еберхарт запропонували новий алгоритм оптимізації нелінійних функцій, що було названо методом рою часток. Алгоритм було створено на базі алгоритму Boids. Розроблений ними алгоритм був також достатньо простий. Правила алгоритму наступні: кожна частина зграї рухається до оптимального розв'язку (їжі), обмінюючись інформацією з сусідніми частинками.

Початковий алгоритм працює з населенням (бджолиним роєм) кандидатів розв'язків (частки). Рух відбувається до двох основних позицій: до найкращої позиції самої частки та найкращої позиції всього рою. Цей процес повторюється ітеративно. Тому коли позиція рою стає кращою, то кожна частинка спрямовує свій рух туди.

Нехай S – число частинок в рою, кожна частинка має положення $x_i \in R_n$ і швидкість $v_i \in R_n$. Нехай p_i – найкраща позиція частинки i, а g – найкраще положення всього рою.

Алгоритм рою часток створено відносно недавно (в 1995 році), але деякі вчені вже запропонували модифікації цього алгоритму. Нові роботи з модифікації алгоритму публікуються і по сьогодення.

Тоді маємо псевдокод алгоритму (Рис. 1.6.1.1).

Одна з модифікацій алгоритму отримала назву LBEST. Ця модифікація була опублікована самими Кеннеді та Ебархартом. Оригінальний алгоритм вони назвали GBEST – "global best", бо він орієнтується на глобальний найкращий розв'язок. Модифікація отримала

назву LBEST, бо розшифровується як “local best”. При оновленні ітеративних параметрів частки, вона використовує інформацію про сусідні частки, а не найкращі параметри всього рою. Сусідні вважаються частки, індекс яких відрізняється на одиницю. Він є більш повільним за оригінальний алгоритм, але уникає локальних екстремумів.

```
for i = 1, ..., S
  Initialize the particle's position x_i
  Initialize the particle's best known position to its initial position:
p_i ← x_i
  if f(p_i) < f(g)
    update the swarm's best known position g ← p_i
  end if
  Initialize the particle's velocity v_i
end for
while a termination criterion is not met do:
  for i = 1, ..., S
    for d = 1, ..., n
      Pick random numbers: r_p, r_g ∈ [0; 1]
      Update the particle's velocity v_{i,d}
    end for
    Update the particle's position x_i
    if f(x_i) < f(p_i)
      Update the particle's best known position p_i ← x_i
      if f(p_i) < f(g)
        Update the swarm's best-known position g ← p_i
      end if
    end if
  end for
end while
```

Рисунок 1.6.1.1 – Псевдокод алгоритму рою частинок

Наступна модифікація була запропонована вже в 1998 році Юхі Ші та Расселом Еберхартом. В своїй роботі вчені помітили, що головною проблемою при вирішенні задачі глобальної оптимізації є вагання між детальним дослідженням простору та кількістю кроків алгоритму. В залежності від типу задачі цей баланс повинен бути різним, тобто динамічним. Додавши до вектору зміни швидкості єдиний коефіцієнт названий коефіцієнтом інерції, вчені скоротили час оптимізації та детальніше дослідили простір.

В цій же роботі вчені сказали, що коефіцієнт інерції – це не обов'язково константа, а може бути і змінна, що може змінюватись за будь-яким законом. Тому в 1999 році вони почали використовувати лінійний закон убування.

Досі ведеться багато досліджень стосовно пошуку оптимальних параметрів алгоритму та покращення алгоритму. Великої популярності отримали модифікації, що пропонують використовувати цей алгоритм в поєднанні з іншими.

1.6.2. Алгоритм кажанів

Один з метаевристичних алгоритмів, що моделює поведінку кажанів. Цей алгоритм є підвидом ройового інтелекту. Алгоритм розроблено в 2010 році Янгом. Одна з переваг алгоритму – його швидкість.

Природа завжди була повна таємниць. Це дуже приваблювало різних дослідників. На базі багатьох природніх процесів побудовано тисяча алгоритмів. Алгоритм кажанів не є виключенням. Алгоритм є потенційно більш потужним за його «батька» - алгоритм рою часток.

Кажани є абсолютно унікальними комахоїдними тваринами. Більшість кажанів володіє досконало ехолокацією, котрі вони використовують для пошуку здобичі, пересування в абсолютній темряві. Кажани є єдиними ссавцями, котрі мають крила. Підраховано, що існує приблизно тисячі видів кажанів, що значно відрізняються один від одного.

Їх розміри коливаються від крихітних (миші масою до 1.5-2 грами) до гігантських кажанів (маса 2 кілограми), розмах крил котрих дорівнює приблизно двом метрам [5].

Ехолокація – це спосіб визначення положення предметів в просторі за допомогою часу затримки відбиття хвилі звуку. Існує декілька підвидів ехолокації: якщо хвиля є звуковою, то цей вид називається звуколокація, а якщо хвиля є радіохвилею – радіолокація.

Ехолокація була відкрита в 18 столітті італійським вченим Ладзаро Спалланцані. Він досліджував поведінку кажанів. Ладзаро помітив, що кажани пересуваються в абсолютній темряві без будь-яких перешкод і з легкістю уникають предметів, що не вдається навіть совам. В своєму досліді він вдався навіть до того, що осліпив кажанів. Але навіть і після цього вони літали на рівні зі зрячими кажанами. Колега Ладзаро провів інший експеримент – він заліпив вуха кажанів воском. Після цього вони стикалися з усіма предметами. Звідси вчені зробили висновок – кажани орієнтуються за допомогою слуху. Ця ідея в той час була висміяна.

Вперше думки про активне використання ехолокації кажанами були озвучені лише в 1912 році. Вчені припустили, що кажани використовують низькочастотні хвилі з частотою 15 Гц.

Підтвердження цьому факту знайшлося ще пізніше – у 1938 році завдяки вченим Гріффіну та Пірсу. Саме Гріффін і запровадив термін «ехолокація». Ехолокацією користуються не лише кажани, а і дельфіни, тюлені та деякі види птахів. Людина також користується ехолокацією – почувши звук в приміщенні, людина може оцінити приблизні розміри цього самого приміщення.

Кажани користуються ехолокацією для того, щоб полювати, уникати перешкод та знайти ночівлю в темряві. Вони видають гучний звуковий імпульс і слухають відлуння, що відбивається від оточуючих предметів. Цей гучний звуковий імпульс людина не може почути. Кажани використовують декілька видів цього імпульсу, що відрізняються між

собою характеристиками звуку. Від виду імпульсу залежить навіть стратегія полювання кажанів [6].

Малі кажани використовують ехолокацію для того щоб виживати, в той час великі кажани можуть взагалі не користуватися нею. Малі кажани можуть омивати будь-якого розміру перешкоду, користуючись ехолокацією, навіть якщо розміри перешкоди менші за волосся людини.

Кожен імпульс дуже короткий тривалістю приблизно 10 мс. Діапазон частот звукових імпульсів, що видає більшість кажанів знаходиться приблизно в районі 25 кГц – 100 кГц. Деякі з кажанів можуть видавати звуковий імпульс з частотою до 150 кГц. При полюванні за здобиччю, імпульси можуть випромінюватись зі швидкістю до 200 імпульсів в секунду.

Алгоритм кажанів використовує наступні правила:

- Всі кажани використовують ехолокацію щоб аналізувати відстань, а також відрізняти здобич і перешкоди
- Кажани переміщуються випадковим чином з певною швидкістю v_i з фіксованою частотою f_{min}, змінною довжиною хвилі звуку λ, з певною гучністю A_0, щоб знайти здобич. Вони самі регулюють довжину хвилі звуку.
- Гучність звуку змінюються починаючи з максимального значення A_0 до мінімального фіксованого значення A_{min}.

Швидкість звуку в повітрі дорівнює приблизно 300 м/с. Тому довжина хвилі для звуку з постійною частотою f задається формулою.

$$\lambda = \frac{v}{f} \tag{1}$$

Саме тому звукові імпульси з частотою від 25 кГц до 150 кГц мають довжину хвилі в діапазоні від 2 мм до 12 мм.

Гучність звукових імпульсів, що використовують кажани, дорівнює понад 110 децибел. Людське вухо їх не помічає оскільки ці хвилі є ультразвуком.

Вивчення кажанів показало, що малі кажани використовують затримку від випромінювання сигналу, до виявлення відлуння, різницю в часі виявлення відлуння в двох вухах для того щоб побудувати тривимірну модель оточуючого простору. За цими даними вони можуть виявити відстань до оточуючих предметів, відстань до цілі, тип здобичі, швидкість її руху, навіть якщо здобиччю є дуже дрібна комаха. Кажани в змозі це виявити завдяки ефекту Допплера.

Ефект Допплера – це зміна частот і довжини хвилі випромінення через рух джерела випромінення хвилі. Ефект названо в честь австрійського фізика Крістіана Допплера.

Якщо джерело хвиль рухається в середовищі і при цьому випромінює хвиль, то відстань між хвилями залежить від швидкості та напрямку руху джерела і приймача. Якщо джерело рухається в напрямку до приймача, тобто наздоганяє хвилі, то довжина хвилі зменшується і навпаки – якщо рухається в напрямку протилежному джерелу, то довжина збільшується.

$$\lambda = \frac{2\pi(c - v)}{w_0} \tag{2}$$

де w_0 – кутова частота хвилі, c – швидкість поширення хвиль в середовищі, v – швидкість джерела звуку відносно середовища (зі знаком «+», якщо джерело наближується до приймача і зі знаком «-», якщо віддаляється).

Для спрощення алгоритму будемо використовувати наступні наближення: частоти хвилі лежать у діапазоні $[f_{min}; f_{max}]$ і відповідають діапазону довжин хвиль $[\lambda_{min}; \lambda_{max}]$.

Опишемо рух кажанів деякими формулами, що необхідні алгоритму.

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{3}$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i \tag{4}$$

$$x_i^t = x_i^{t-1} + v_i^t \quad (5)$$

де $\beta\epsilon[0,1]$ – випадкова величина. В цих формулах x_* - найкращий поточний розв'язок.

Крім того, амплітуда імпульсу і його емісія повинні бути оновленні кожну ітерацію алгоритму. Зазвичай як кажан наближується до цілі, то гучність його імпульсів зменшується, а частота імпульсів збільшується

$$A_i^{t+1} = \alpha A_i^t \quad (6)$$

$$r_i^{t+1} = r_i^0[1 - \exp(-\gamma t)] \quad (7)$$

де α та γ константи. Для будь-яких $0 < \alpha < 1$ та $\gamma > 0$ маємо

$$A_i^t \to 0, r_i^t \to r_i^0, as\ t \to \infty \quad (8)$$

Складність такого алгоритму напряму залежить від кількості осіб в зграї, що використовується в алгоритмі, кількості екстремумів та розмірності задачі.

Звідси отримуємо псевдокод алгоритму (Рис. 1.6.2.1):

```
Objective function f(x), x = (x_1, ..., x_n)
Initialize the bat population x_i (i = 1, 2, ..., n) and v_i
Define pulse frequency f_i at x_i
Initialize pulse rates r_i and the loudness A_i
while (t < Max number of iterations)
  Generate new solutions by adjusting frequency
  Update velocities and locations/solutions
  if (rand > r_i)
    Select solution among best solutions
    Generate a local solution among the selected best solution
  end if
  if (rand < A_i && f(x_i) < f(x_*))
    Accept the new solutions
    Increase r_i and reduce A_i
  end if
  Rank the bats and find the current best x_*
  end while
Postprocess results.
```

Рисунок 1.6.2.1 – Псевдокод алгоритму кажанів

1.6.3. Алгоритм світлячків

Цей алгоритм моделює поведінку світлячків та використовує їх у якості моделі. Вперше алгоритм був опублікований в 2009 році. Це ще один з підвидів ройових алгоритмів. Дуже активно розвивається на сьогоднішній день.

Для дослідження алгоритму було обрано задачу глобальної оптимізації. Дослідженням займалися декілька відомих вчених: Розенброк, Растрігін, Де Джонг. Як і всі ройові алгоритми, цей базується на поведінці рою світлячків.

В основі дослідження лежить світло, що випромінюють світлячки. Це світло є засобом спілкування та комунікації світлячків в їх зграї. За допомогою цього світла вони приваблюють світлячків іншої статті,

повідомляють про небезпеку і її наближення. Менш «привабливі» світлячки переміщуються до більш «привабливих», а якщо світлячок не бачить більш яскравого, то він переміщується випадковим чином. Тобто кожний світлячок характеризується його позицією в просторі та яскравістю [7].

Алгоритм світлячків користується цією інформацію. Тому він побудований на базі наступних правил:

1. Усі світлячки є безстатевими. Тобто один світлячок взаємодіє з іншими незалежно від статі;
2. Привабливість світлячка пропорційна його яскравості. Тобто це означає, що у випадку пари світлячків, менш яскравий буде рухатися до більш яскравого. Так як яскравість світлячків пропорційна привабливості, то ця величина зменшується обернено пропорційна до відстані між парою;
3. Значення функції, що оптимізується обернено пропорційне яскравості світлячка.

Спочатку світлячки розміщуються випадковим чином в просторі. Далі розраховується яскравість світлячків за наступною формулою:

$$I = \frac{1}{f(x) + 1} \tag{1}$$

де $f(x)$ – значення цільової функції.

Відповідно до фізичного закону, яскравість світла залежно від відстані до джерела змінюється за формулою

$$I(r) = \frac{I_0}{r^2} \tag{2}$$

Формулу (2) з урахуванням коефіцієнта поглинання можна подати у Гаусовій формі

$$I(r) = I_0 e^{-\gamma r^2} \tag{3}$$

де I_0 – яскравість світла джерела випромінювання, γ – коефіцієнта поглинання світла, який може залежати від відстані [8].

Відстань $r_{i,j}$ між світлячками i та j обчислюється за формулою евклідової відстані

$$r_{i,j} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} \left(x_{i,k} - x_{,k}\right)^2} \quad (3)$$

де d – вимірність простору.

Для двовимірного простору формула (2) набуває вигляду

$$r_{i,j} = \sqrt{\left(x_i - x_j\right)^2 - \left(y_i - y_j\right)^2} \quad (4)$$

Якщо яскравість першого світлячка менша за яскравість другого, то перший рухається в напрямку до другого за наступною формулою:

$$x_i^{t+1} = x_i^t + \alpha c + \beta(x_i^t - x_j^t) \quad (5)$$

$$\beta = \beta_0 * e^{-\gamma r^2} \quad (6)$$

де γ – коефіцієнт поглинання середовищем світла, r – відстань між світлячками, x_i^t – початкова відстань світлячка i, β – привабливість світлячка j для світлячка i, β_0 – привабливість світлячка за умови відсутності відстані між ними, c – випадкова константа на проміжку $[0; 1]$, α – випадкова величина.

З вищенаведених тверджень отримуємо псевдокод алгоритму (Рис. 1.6.3.1):

Objective function $f(x), x = (x_1, \ldots, x_n)$

Initialize the firefly positions x_i $(i = 1, 2, \ldots, n)$

Define attractiveness β_{ij}

Define best position g

while (t < Max number of iterations)

Update firefly positions

Update attractiveness

Sort fireflies by attractiveness

Find new best position g

end while

Postprocess results.

Рисунок 1.6.3.1 – Псевдокод алгоритму рою світлячків

Алгоритм є дуже простим в реалізації, але має свої недоліки такі як неточність та несталість результату.

Алгоритм рою світлячків використовується в багатьох сферах сьогодні. Дискретний варіант алгоритму (Discrete Firefly Algorithm), який запропонували Саяді та Ремезенін може вирішувати NP-складні проблеми планування. Багатоцільовий алгоритм світлячків (Multipurpose Firefly Algorithm) дуже гарно вирішує проблему відправки чого-небудь до декількох отримувачів. Лагранжевий метод рою часток (Lagrange Firefly Algorithm) вирішує проблеми оптимізації екосистеми [9].

1.7. Обґрунтування теми магістерської дисертації

В якості об’єкту дослідження був обраний алгоритм квіткового запилення так як цей алгоритм є дуже простим в реалізації, має малу кількість вхідних параметрів, алгоритм є малодослідженим та легко може бути модифікований. Алгоритм розв’язує задачу однокритеріальної оптимізації достатньо точно і швидко.

Використовуючи відомі на сьогоднішній день методи зведення задачі багатокритеріальної оптимізації до однокритеріальної, алгоритм без додаткової інформації може виконати і цю задачу.

2. АЛГОРИТМ КВІТКОВОГО ЗАПИЛЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМІЗАЦІЇ

2.1. Алгоритм квіткового запилення для розв'язання задачі однокритеріальної оптимізації

Алгоритм квіткового запилення є однією з найсучасніших метаевристик, що застосовується для розв'язання задачі оптимізації у неперервному просторі.

Запропоновано його було у грудні 2013 року Ксін-Ше Янгом-старшим науковцем національної фізичної лабораторії, що відомий як розробник багатьох евристичних алгоритмів. Він створив такі евристичні алгоритми як алгоритм світлячків в 2008 році, алгоритм зозулі у 2009 році, алгоритм кажанів у 2010 році. Сьогодні ці алгоритми і їх модифікації використовуються у інструментах штучного інтелекту, нейронних мережах та певних інженерних додатках [10].

Алгоритм базується на природному процесі квіткового запилення. На сьогоднішній день досліджено, що існує понад пів мільйона різних видів рослин. Серед них найбільшу кількість становлять насінні рослини – приблизно 300000 видів рослин цього типу. Рослини розмножуються двома шляхами: статевим і безстатевим. Серед форм безстатевого розмноження рослин найпоширенішим є вегетативне розмноження. До таких рослин відносять мохоподібні рослини, сосудисто-спорові та інші.

80% цих рослин розмножується шляхом запилення. В дослідженні важливі дві форми запилення по відношенню рослин до запилювачів – абіотична та біотична. Майже всі рослини (90%) розмножуються шляхом біотичного запилення. Біотична форма запилення – це запилення за допомогою комах, птахів, кажанів. Абіотична форма запилення – це запилення за допомогою абіотичних чинників середовища (повітря, вода).

Крім того існує два види запилення рослин до рослин – самозапилення та перехресне запилення.

При біотичному запилення квітки-запилювачі обирають квітку для запилення враховуючі різні показники: «привабливість», відстань до квітки, висота квітки та багато інших. Тобто найкраща квітка за цими показниками має найбільшу ймовірність запилення. За абіотичного запилення процес контролюється напрямком та швидкістю вітру, швидкістю течії води, тобто квітка обирається майже випадковим чином.

На цьому і базується метод квіткового запилення. Якщо біологічні факти інтерпретувати математичною мовою, то маємо наступне: існує дві гілки пошуку оптимального значення функції. Перша гілка відповідає пошуку наступного оптимального значення функції, базуючись на найкраще відоме значення на даний момент. Друга гілка відповідає пошуку оптимального значення, базуючись на значенні випадкової величини. Переключення між першою і другою гілками алгоритму контролюється випадковою величиною.

З вищенаведених досліджень та наукових фактів розроблено алгоритм квіткового запилення. Алгоритм описано лише трьома правилами [10]:

1. Біотичне запилення та перехресне запилення вважається глобальним запиленням, де рух запилювачів підпорядковується математичним законам;
2. Абіотичне запилення та самозапилення вважається локальним запиленням;
3. Переключення між локальним і глобальним запилення контролюється випадковою величиною $p \in [0; 1]$.

Якщо також вважати, що квітки, як і запилювачі представляють множину векторів допустимих змінних функції $f(x)$, то можна розробити математичний алгоритм пошуку оптимального значення функції оптимізації. Множина допустимих значень аргументу є «полем квіток», які

можуть бути запиленні, а значення аргументу є позиція запилювача в цьому полі.

На сьогоднішній день вже створено багато модифікацій алгоритму квіткового запилення. Понад половини всіх модифікацій – це гібридизація алгоритму з іншими відомими методами. Понад чверть досліджень присвячене модифікації оригінального алгоритму квіткового запилення. Майже чверть досліджень алгоритму – це дослідження оптимальних параметрів алгоритму. Інтерес до алгоритму квіткового запилення за останні п’ять років його існування тільки збільшився. [11]

Модифікації та гібридизації алгоритму описані в таких популярних наукових системах як IEEE Explorer (база даних пошуку та досліджень різних журнальних статей, конференцій, технічних стандартів), SpringerLink, Taylor & Francis (книжне видавництво), ScienceDirect (база даних наукових публікацій) та інших. Станом на 2018 рік, алгоритм має понад 150 публікацій, що пов’язанні з модифікацією та гібридизацією. Найбільшу кількість публікацій було оприлюднено у 2016 році (81 публікація) [11].

Найбільша кількість модифікацій алгоритму пов’язана з збільшенням пошукової здібності алгоритму та зменшенням кількості параметрів алгоритму.

Найпопулярнішою гібридизацією алгоритму є схрещені алгоритм квіткового запилення та алгоритмом імітації відпалу Flower Pollination Simulated Annealing algorithm (FPSA). Також вчені запропонували алгоритм квіткового запилення схрещений з алгоритмом K-Means для кластеризації даних. Серед найпопулярніших гібридів квіткового запилення схрещеного з іншим метаевристичним алгоритмом є алгоритм квіткового запилення схрещений з алгоритмом рою часток Flower Pollination Particle Swarm Optimization Algorithm (FPPSO).

Алгоритм застосовується сьогодні в багатьох сферах. Алгоритм використовують в сфері електроенергії (52%), передачі та класифікації

даних (11%), обробки зображень та сигналів (9%) [12]. В майбутньому алгоритм планують застосовувати ще для розв'язання складних комбінаторних задач.

2.2. Загальний принцип роботи алгоритму

Для початку роботи алгоритму треба випадковим чином ініціалізувати запилювачів, тобто встановити їм початкову позицію. Серед цих запилювачів треба знайти такого, за якого функція приймає мінімальне/максимальне значення. На даний момент - це позиція найкращої квітки, на якій знаходиться запилювач. Деякі запилювачі будуть прямувати до найкращого запилювача, інші – до іншого випадкового запилювача [10].

Перше правило запилення (біотичне запилення) описується формулою

$$x_i^{t+1} = x_i^t + L(x_i^t - g_*) \tag{1}$$

де x_i^t- i-й запилювач або вектор поточного розв'язку $f(x)$ на кроці t, g_*- кращий розв'язок, L – випадкова послідовність яка підпорядковується розподілу Леві. Розподіл Леві є неперервним розподілом ймовірностей для невід'ємної випадкової величини

$$L \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}} \tag{2}$$

де λ, s – параметри алгоритму, $\Gamma(\lambda)$ – гамма-функція.

Друге правило запилення (абіотичне запилення) можна описати формулою

$$x_i^{t+1} = x_i^t + \epsilon(x_j^t - x_k^t) \tag{3}$$

де x_i^t- i-й запилювач або вектор поточного розв'язку $f(x)$ на кроці t, x_j^t- j-й запилювач або вектор поточного розв'язку $f(x)$ на кроці t, x_k^t- k-й запилювач або вектор поточного розв'язку $f(x)$ на кроці t, ϵ – випадкова

послідовність яка підпорядковується рівномірному розподілу на проміжку [0; 1]. Номера k та j обираються випадковим чином [13].

Неперервний розподіл – це неперервний розподіл ймовірностей для невід'ємної випадкової величини

$$\epsilon = \begin{cases} 0, x < a \\ \dfrac{x-a}{x-b}, a \leq x < b \\ 1, x \geq b \end{cases} \quad (4)$$

Перше правило відповідає глобальному запиленню, друге – локальному.

З врахуванням наведених вище формул, маємо псевдокод алгоритму (Рис. 2.2.1).

Initialize a population with n flowers/pollen gametes with random solutions
Find the best solution g in the initial population
Define a switching probability p ∈ [0, 1]
while (t < MaxGeneration)
for i = 1 : n (all n flowers in the population)
if rand < p,
Draw a (d-dimensional) step vector L which obeys Levy distribution
Group global pollination via $\boldsymbol{x_i^{t+1} = x_i^t + L(g_* - x_i^t)}$
else
Draw ϱ from a uniform distribution in [0,1]
Randomly choose k among group solutions
Do group local pollination via $\boldsymbol{x_i^{t+1} = x_i^t + Q(x_k^t - x_i^t)}$
end if
Evaluate new solutions
If new solutions are better, update them in the population
end for
end while
Find global best solution g* among all groups

Рисунок 2.2.1 – Псевдокод алгоритму квіткового запилення

Перевагами алгоритму є його простота реалізації та мінімальна кількість гілок алгоритму – лише дві. Але оригінальний алгоритм має і певну кількість недоліків.

Один з головних недоліків є те, що якщо найкращий на даний момент запилювач «застрягне» у локальному екстремумі, то у випадку якщо $p > 0.5$ велика кількість запилювачів буде прямувати до нього і також застрягне у цій позиції. У випадку якщо $p < 0.5$, то запилювачі будуть рухатися хаотично один до одного і найкращий запилювач може попрямувати до найгіршого, тим самим спрямувавши всіх запилювачів до гірших розв'язків.

Іншим недоліком є те, що всі запилювачі прямують лише до одного найкращого запилювача, що дуже зменшує досліджувану область та пошукові можливості алгоритму. Це також призводить до застрягання всіх запилювачів в локальному екстремумі.

Ще одним недоліком є неточність вибору наступної позиції запилювача за другим правилом. За другим правилом запилювач змінює свою позицію на відстань, що є пропорційною до відстані між двома випадковими запилювачами. Під час випробовувань встановлено, що майже у всіх випадках нова позиція є гіршою за попередню позицію.

2.3. Порівняння алгоритму з іншими метаевристичними алгоритмами оптимізації

Алгоритм квіткового запилення є найсучаснішим метаевристичним алгоритмом, бо його було розроблено у 2013 році. Останнім перед алгоритмом квіткового запилення було розроблено алгоритм кажанів у 2010 році. Алгоритм в порівнянні з його попередниками є найпростішим у реалізації та його створено базуючись лише на двох правилах. Наприклад, алгоритм кажанів ще базується на фізичному явищі ефекті Допплера, алгоритм світлячків – фізичному законі поширенні світла в середовищі.

Реалізація алгоритму квіткового запилення є дуже схожою на реалізацію алгоритму рою часток, хоча вони і розроблені на базі двох різних явищ в природі. Алгоритм квіткового запилення є покращеною версію алгоритму рою часток з математичної точки зору. Алгоритм рою часток залежав не тільки від позиції найкращої частки, але і від швидкості її переміщення. Для алгоритму квіткового запилення кожен запилювач може переміститися в будь-яку точку за один крок алгоритму, що значно прискорило алгоритм. У той же час, це збільшило ймовірність «застрягання» запилювачів.

Також алгоритм рою часток залежить ще від певної групи параметрів, що обираються користувачем. В алгоритмі квіткового запилення єдиною константою, що встановлюється користувачем є ймовірність переключення між локальним і глобальним запиленням.

Через те, що під час вибору нової позиції в алгоритмі рою часток виконується більша кількість операцій, алгоритм квіткового запилення є більш швидким за його попередника.

На сьогоднішній день існують версії алгоритму, що одночасно покращують і ускладнюють його, використовуючи алгоритм рою часток, бо кожен з алгоритмів має переваги та недоліки в порівнянні один з одним.

2.4. Розробка удосконаленого алгоритму квіткового запилення

З вищеназваних недоліків наслідками обох є те, що вся група запилювачів може застрягнути в зоні локального екстремуму.

Тому було прийнято рішення поділити запилювачів на певну кількість незалежних груп, кожна з яких не знає нічого про існування інших груп. В кожній групі буде існувати кращий запилювач – значення функції якого мінімальне/максимальне в групі. До нього буде прямувати вся група запилювачів. Таким чином в випадку застрягання однієї групи запилювачів завжди буде існувати інша альтернативна група запилювачів.

Оптимальний розв’язок буде обиратися серед кращих запилювачів кожної групи.

Тим самим перше правило алгоритму оновлюються до наступного вигляду

$$x_{i,j}^{t+1} = x_{i,j}^{t} + L(x_{i,j}^{t} - g_{j*}) \quad (1)$$

де $x_{i,j}^{t}$- i-й запилювач групи j або вектор поточного розв’язку $f(x)$ на кроці t, g_{j*}- кращий розв’язок групи j, L – випадкова послідовність яка підпорядковується розподілу Леві.

Друге правило алгоритму має наступний вигляд

$$x_{i,j}^{t+1} = x_{i,j}^{t} + \epsilon(x_{k,j}^{t} - x_{m,j}^{t}) \quad (2)$$

де $x_{i,j}^{t}$- i-й запилювач групи j або вектор поточного розв’язку $f(x)$ на кроці t, $x_{k,j}^{t}$- k-й запилювач групи j або вектор поточного розв’язку $f(x)$ на кроці t, $x_{m,j}^{t}$- m-й запилювач групи j або вектор поточного розв’язку $f(x)$ на кроці t, ϵ – випадкова послідовність яка підпорядковується рівномірному розподілу на проміжку [0; 1]. Номера k та m обираються випадковим чином [14].

При оптимальному виборі кількості груп та кількості запилювачів в кожній з групи, ця модифікація значно розширює зону пошуку глобального екстремуму. Також в ході випробовувань було помічено, що при збільшені кількості груп точність знайденого екстремуму значно швидше збільшується, ніж при збільшенні кількості запилювачів в групі.

При застосування цієї модифікації ймовірність застрягання всіх груп запилювачів зменшується. Для подальшого покращення прийнято рішення застосувати ще одну модифікацію - скидання значення випадкового запилювача з постійною дуже малою ймовірністю до випадкового значення. В цьому випадку запилювач у випадку застрягання у локальному екстремумі буде повернутий до початкової позиції і матиме шанс вивести

групу запилювачів з локального екстремуму. Це дає змогу оминути цей локальний екстремум у майбутніх рухах запилювачів.

За другим правилом запилення запилювач змінює свою позицію на відстань, що пропорційна відстані між двома випадковими запилювачами. Насправді, ця величина є випадковою величиною і тому якщо наступна позиція запилювача обиралася за другою гілкою запилення, то наступна позиція зазвичай була гіршою за попередню.

Тому прийнято рішення змінити формулу на більш схожий вигляд до правила глобального запилення.

$$x_{i,j}^{t+1} = x_{i,j}^{t} + \epsilon(x_{i,j}^{t} - x_{k,j}^{t}) \quad (3)$$

де $x_{i,j}^{t}$- i-й запилювач групи j або вектор поточного розв'язку $f(x)$ на кроці t, $x_{k,j}^{t}$- k-й запилювач групи j або вектор поточного розв'язку $f(x)$ на кроці t, ϵ – випадкова послідовність яка підпорядковується рівномірному розподілу на проміжку [0; 1]. Номер k обирається випадковим чином.

В даній модифікації глобальне та локальне запилення відрізняться тим, що замість наближення до кращого розв'язку, наближення відбувається до випадкового розв'язку. Це виключає ймовірність випадкового переміщення запилювача, що зазвичай тільки погіршувало роботу оригінального алгоритму.

З урахуванням всіх модифікацій маємо наступний псевдокод алгоритму (Рис. 2.4.1).

Initialize a population of m groups with n flowers/pollen gametes with random solutions

Find the best solution gj in the initial population among the groups

Define a switching probability p ∈ [0, 1]

Define a reset probability preset ∈ [0, 1]

while (t < MaxGeneration)

for j = 1 : m (all m flower groups)

for i = 1 : n (all n flowers in the population)

if rand < p,

Draw a (d-dimensional) step vector L which obeys Levy distribution

Group global pollination via $x_{j,i}^{t+1} = x_{j,i}^{t} + L(g_{j*} - x_{j,i}^{t})$

else

Draw ϱ from a uniform distribution in [0,1]

Randomly choose k among group solutions

Do group local pollination via $x_{j,i}^{t+1} = x_{j,i}^{t} + Q(x_{k,i}^{t} - x_{j,i}^{t})$

end if

Evaluate new solutions

If new solutions are better, update them in the population

If rand < preset

Randomly choose k among flowers

Reset $x_{j,i,k}$**(group j, flower i, root k)**

end for

Find the current group best solution gj*

end for

end while

Find global best solution g* among all groups

Рисунок 2.4.1 – Псевдокод модифікованого алгоритму квіткового запилення

2.5. Порівняння оригінального і модифікованого алгоритмів квіткового запилення при розв’язанні задачі однокритеріальної оптимізації

В порівнянні з оригінальним алгоритмом, модифікований алгоритм має лише 3 зміни: введення груп запилювачів, зміна правила локального (абіотичного) запилення та «скидання» запилювача. Ці зміни значно покращили пошукові здібності алгоритму та швидкість знаходження мінімуму/максимуму функції оптимізації.

Введення групи запилювачів дозволило зробити пошук екстремуму функції в ширшій області, ніж це робив оригінальний алгоритм. Хоча на перший погляд здається, що додавання груп запилювачів призводить до збільшення кількості запилювачів, а отже і до швидкості роботи алгоритму, але насправді це не так. При введені поняття «група» можна зменшити кількість запилювачів в групі, наприклад, якщо для оригінального алгоритму використовується 25 запилювачів, то для модифікованого достатньо 5 груп по 5 запилювачів, а не по 25 запилювачів. В такому випадку один запуск модифікованого алгоритму це є 5 паралельних запусків оригінального алгоритму.

Наступна модифікація, а саме введення поняття «скидання» запилювача, дозволила виводити групи запилювачів з зон локального екстремуму, що збільшило пошукові здібності алгоритму. Запилювачі встигають потрапити у локальний екстремум, але потім вийти з нього через «скидання» одного з запилювачів. Але ймовірність скидання повинна залишатися достатньо малою, бо тоді запилювачі будуть часто губити найкращі позиції і повертатися до стартової позиції. Тому використовувалась ймовірність, що дорівнювала приблизно 1e^(-4). Через велику кількість ітерацій, запилювачі встигають покидати зони локальних екстремумів досить часто.

Останнє правило покращило лише локальне (абіотичне) запилення, але тим самим зробило рух запилювачів спрямованим до іншого

запилювача. В оригінальному алгоритмі через спрямування руху випадковим чином в другій гілці алгоритму, призвело до того, що запилювачі покидали кращі позиції. Звичайно за модифікованим другим правилом запилювачі іноді покидають кращі позиції, але іноді це допомагає їм покинути зони локального екстремуму в пошуках нових областей.

Всі три модифікації спрямованні на ігнорування локальних екстремумів та збільшення пошукових здібностей алгоритму. Вони є простими в реалізації та не потребують додаткових апаратних витрат. Випробовування показали, що ці модифікації покращують роботу алгоритму за час, що не перевищує час пошуку екстремуму за допомогою оригінального алгоритму квіткового запилення.

2.6. Вдосконалення алгоритму квіткового запилення для розв’язання задачі багатокритеріальної оптимізації

Метод квіткового запилення розроблено для розв’язання задачі однокритеріальної оптимізації. Користуючись методом скаляризації, алгоритм квіткового запилення можна модифікувати до розв’язання задачі багатокритеріальної оптимізації. Для цього вектор критеріїв необхідно попередньо скаляризувати до єдиного критерію та виконати однокритеріальну оптимізацію цього критерію, застосувавши метод квіткового запилення.

Реалізація алгоритму залишається незалежною від кількості вхідних критеріїв, бо в випадку скаляризації, всі вхідні критерії будуть зведені до одного. В цій модифікації алгоритм все ще зданий вирішувати задачу однокритеріальної оптимізації, що для алгоритму є лише частковим випадком задачі багатокритеріальної оптимізації з одним лише критерієм.

В випадку скаляризації, знайдений розв’язок буде одним з розв’язків оптимальних за Парето, тобто розв’язок з множини Парето.

В цьому випадку матимемо наступний псевдокод алгоритму (Рис. 2.6.1) [15].

З псевдокоду видно, що для того щоб розв’язувати багатокритеріальні задачі оптимізації, необхідно було лише додати цикл та приведення функцій оптимізації до одного критерія. Це є перевагою алгоритму.

Initialize a population of m groups with n flowers/pollen gametes with random solutions
Find the best solution gj in the initial population among the groups
Define a switching probability p ∈ [0, 1]
Define a reset probability preset ∈ [0, 1]
for k = 1 : N (points number on Pareto front)
  Generate wk where $\sum_{k=1}^{N} w_k = 1$
  Generate one function $f = \sum_{k=1}^{N} w_k f_k$
    while (t < MaxGeneration)
      for j = 1 : m (all m flower groups)
        for i = 1 : n (all n flowers in the population)
          if rand < p,
            Draw a (d-dimensional) step vector L which obeys Levy distribution
            Group global pollination via $x_{j,i}^{t+1} = x_{j,i}^{t} + L(g_{j*} - x_{j,i}^{t})$
          else
            Draw ϱ from a uniform distribution in [0,1]
            Randomly choose k among group solutions
            Do group local pollination via $x_{j,i}^{t+1} = x_{j,i}^{t} + Q(x_{k,i}^{t} - x_{j,i}^{t})$
          end if
        Evaluate new solutions
        If new solutions are better, update them in the population
        If rand < preset
          Randomly choose k among flowers
          Reset $x_{j,i,k}$**(group j, flower i, root k)**
      end for
      Find the current group best solution gj*
    end for
  end while
Find global best solution g* among all groups
end for

Рисунок 2.6.1 – Псевдокод модифікованого алгоритму квіткового запилення для розв’язання задачі багатокритеріальної оптимізації

2.7. Порівняння оригінального алгоритму та модифікованого при розв’язанні задачі багатокритеріальної оптимізації

Так само як і модифікований алгоритм, оригінальний алгоритм квіткового запилення легко вдосконалити до алгоритму багатокритеріальної оптимізації функцій.

Головною проблемою оригінального алгоритму є «застрягання» в зоні локального екстремуму функції. У випадку скаляризації вхідних критеріїв оптимізації, оригінальний алгоритм оптимізує єдиний скаляризований критерій оптимізації.

Якщо найкращий розв’язок «застрягне» у локальному екстремумі алгоритму, кожен з початкових показників оптимізації прийме це розв’язок, що не є оптимальним за Парето. Також треба пам’ятати, що знайдений оптимальний розв’язок скаляризованої функції буде поширений на всі критерії. А це означає, що похибка знайденого розв’язку буде зростати з збільшенням кількості критеріїв.

В оригінальному алгоритмі не передбачено інструментів виходу запилювачів з локального мінімуму/максимуму, що є в модифікованому алгоритмі квіткового запилення. Через це похибка знайденого розв’язку за допомогою модифікованого алгоритму квіткового запилення буде набагато меншою за похибку знайденого розв’язку за допомогою оригінального алгоритму.

Псевдокод оригінального алгоритму для розв’язання задачі багатокритеріальної оптимізації наступний (Рис. 2.7.1).

Initialize a population with n flowers/pollen gametes with random solutions

Find the best solution g in the initial population

Define a switching probability p ∈ [0, 1]

for k = 1 : N (points number on Pareto front)

Generate wk where $\sum_{k=1}^{N} w_k = 1$

Generate one function $f = \sum_{k=1}^{N} w_k f_k$

while (t < MaxGeneration)

for i = 1 : n (all n flowers in the population)

if rand < p,

Draw a (d-dimensional) step vector L which obeys Levy distribution

Group global pollination via $x_i^{t+1} = x_j^t + L(g_* - x_i^t)$

else

Draw ϱ from a uniform distribution in [0,1]

Randomly choose k and m among solutions

Do group local pollination via $x_i^{t+1} = x_i^t + \epsilon(x_k^t - x_m^t)$

end if

Evaluate new solutions

If new solutions are better, update them in the population

end for

end while

Find global best solution g* among all groups

end for

Рисунок 2.7.1 – Псевдокод оригінального алгоритму квіткового запилення для розв’язання задачі багатокритеріальної оптимізації

3. ВИБІР ПРОГРАМНИХ ЗАСОБІВ

3.1. Середовище та компоненти розробки

Visual Studio – це один з основних продуктів розроблених компанією Microsoft, що містить велику кількість інструментів для розробників програмного забезпечення, а також інтегроване середовище розробки програмного забезпечення Integrated Development Environment (IDE). Цей продукт дає можливість розробникам створювати програми багатьма мовами програмування, а саме C#, C, C++, Python, F#, JavaScript та інші. Це середовище розробки дозволяє дає змогу розроблювати консольні додатки, додатки що мають десктопний інтерфейс користувача, серверні додатки, веб-сайти, веб-додатки, додатки що використовують Windows Forms, додатки для платформ Windows, Windows Mobile, Windows Server, Windows CE, .NET Framework, Xbox, Silverlight [16].

Середовище розробки поширюється з багатою кількістю інструментів такими як редактор вихідного коду, відладчик коду, відладчик машинного коду, інструменти для роботи з різними типами баз даних, редактор форм, веб-редактор, дизайнер класів і так далі. Visual Studio дозволяє створювати та підключати різні плагіни, тим самим розширюючи свою функціональність.

Visual Studio складається з наступних компонентів:

- Visual .NET;
- Visual Basic;
- Visual C++;
- Visual C#;
- Visual F#.

В минулому продукт включав в себе також Visual J#, Visual FoxPro, Visual Source Safe та інші компоненти.

Перша версія продукту вийшла в далекому 1995 році та мала назву Visual Studio. Ця версія продукту була доступною лише для фахівців компанії Microsoft.

Перша версія для публічного використання була випущена в січні 1997 року та мала назву Visual Studio 97. Продукт було випущено в двох версіях – Enterprise та Professional. В ній було зібрано сучасні засоби розробки програмного забезпечення, але на той час середовище розробки не включало компонентів для розробки програм мовою C#. Насправді це було лише спробою монополізувати ринок, створивши єдиний засіб для створення програмного забезпечення, що включає в себе засоби для розробки програм різними мовами програмування.

Вже через півтори роки в червні 1998 року вийшла нова версія продукту під назвою Visual Studio 6.0. Це була остання версія продукту, що працювала на платформі Windows 9x. Навіть сьогодні ця версія продукту використовується для підтримки продуктів написаних на «класичній» версії Visual Basic.

В цей самий час компанія Microsoft починає розробку найпопулярнішої на сьогоднішній день платформи для написання додатків Windows різного спрямування – платформу .NET Framework. На її розробку компанія витратила 4 роки та випустила її у складі нової версії Visual Studio .NET. З цього часу середовище розробки почало активно використовуватись як найпопулярніший засіб для програмування додатків платформи Windows.

На сьогоднішній день остання версія продукту була випущена у березні 2017 року і має назву Visual Studio 2017. Середовище розробки підтримує всі версії .NET Framework, починаючи з версії 2.0. Також це єдине середовище розробки, що підтримує розробку додатків на .NET Core. Дана версія продукту випущена у декількох виданнях: Community, Professional, Enterprise, Test Professional, Express.

Visual Studio Code – це простий редактор коду, що розроблений компанією Microsoft для Windows, Linux та MacOS. Головною відмінністю від Visual Studio є те, що це «легкий» редактор коду, що надзвичайно просто розширюється за допомогою плагінів до реального середовища розробки. Це кросплатформений інструмент для розробки веб- і хмарних додатків. Visual Code поширюється безкоштовно для всіх платформ.

Редактор коду VS Code було анонсовано в квітні 2015 року на конференції Build. Незабаром було випущено бета версію продукту. В кінці 2015 року Visual Studio Code отримав ліцензію MIT, вихідний код було опубліковано в GitHub.

На даний момент редактор підтримує багато мов програмування, серед яких C#, F#, C++, C, Python, TypeScript, JavaScript, Ruby On Rails, R, CSS, HTML, LESS, SASS та багато інших. Остання версія Visual Code була випущена в серпні 2018 року.

.NET Framework – програмна платформа, яку випустила компанія Microsoft в 2002 році. Сама платформа – це середовище виконання, що виконує програми написанні різними мовами, аде які підтримують .NET Framework. Об’єктно-орієнтовані та бібліотечні можливості є доступними в будь-якій мові програмування, що підтримує це середовище.

До недавно вважалося, що платформа .NET Framework є відповіддю компанії Sun Microsystems та їх платформі Java. Але це не є правдою, бо платформи .NET Framework та Java навіть з самого початку їх розвитку мали багато відмінностей.

Платформа .NET є технологією, яку компанією Microsoft запатентувала та вона розрахована на роботу під операційною системою Microsoft Windows. Існують інші проекти, що дозволяють запускати програми написанні на .NET Framework під іншими ОС, такими як Linux, MacOS. Це проекти Mono та Portable.NET. На сьогоднішній момент Microsoft продовжує дослідження в галузі розвитку цієї платформи та її поширення на різні системи. Нова платформа отримала назву .NET Core.

Розробка фреймворку почалася в далекому 1999 році, а офіційно опубліковано першу звістку про платформу було у 2000 році, коли Білл Гейтс давав останню прес-конференцію як голова компанії Microsoft. Головною відмінністю від інших платформ було те, що вона розроблювалась для того щоб писати додатки, які гарантовано будуть працювати на всіх Windows-подібних системах.

Основною ідеєю і ціллю розробки .NET Framework було те, що платформа буде давати розробнику необмежені можливості в створенні додатків різного типу та спрямування, що здатні виконуватись на різних типах пристроїв та різних середовищах. Іншим орієнтиром було спрямування на сімейство Windows систем.

Архітектура .NET Framework дуже детально описана в специфікації Common Language Infrastructure (CLI), що є офіційно зареєстрованою специфікацією ISO та ECMA. Найновіша версія стандарту є CLI 3.5 [16].

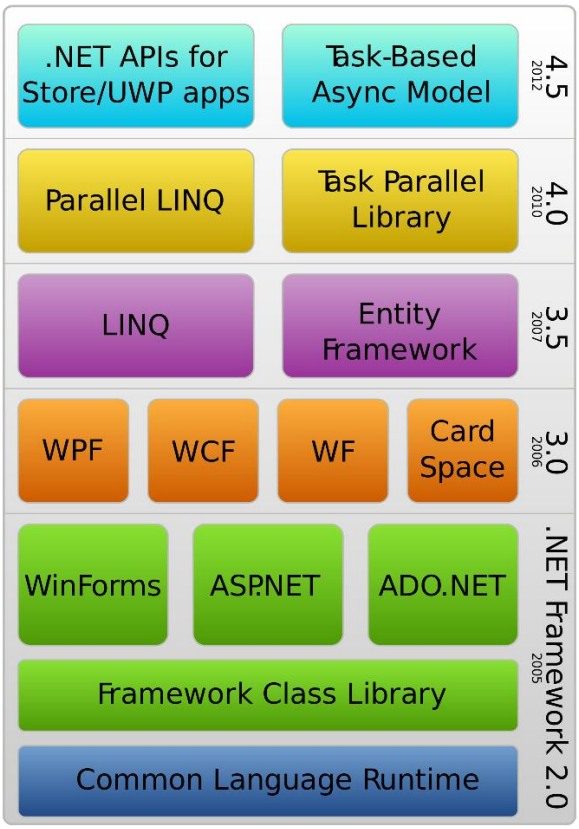

Рисунок 3.1.1 – Стек технологій .NET Framework

Програма розроблена будь-якою мовою, що підтримується .NET Framework, спочатку компілюється у проміжний байт код, що має назву загальна проміжна мова Common Intermediate Language (CLI). Результатом компіляції є збірка (assembly). Наступним кроком є виконання байт коду віртуальною машиною, що має назву загальне середовище виконання Common Language Runtime (CLR).

Використання віртуальної машини є не обов'язковим, але розробники платформи наполегливо рекомендують її використання, бо це гарантує правильне використання апаратних ресурсів машини. В разі використання віртуальної машини код виконується вбудованим компілятором, що має назву Just in time компілятором (JIT). Віртуальна машина сама виділяє необхідну пам'ять, оброблює виключення тим самим позбавляючи програмістів від найбільш важкої роботи.

Об'єктні класи платформи доступні для використання на будь-якій мові програмування, що підтримує платформа. До бібліотеки платформи входять наступні класи:

- Windows Forms;
- ADO.NET;
- ASP.NET;
- Language Intermediate Query (LINQ);
- Windows Presentation Foundation (WPF);
- Windows Communication Foundation (WCF).

Ця бібліотека має назву Framework Class Library (FCL). Ядро FCL називається Base Class Library (BCL).

Наступні середовища розробки підтримують .NET Framework:

- Microsoft Visual Studio;
- SharpDevelop;
- MonoDevelop;
- Zonno;

- JetBrains Rider.

Насправді додатки, що написані мовою програмування яку підтримує .NET Framework, можна писати у будь-якому текстовому редакторі.

Остання версія 4.7.2 платформи була випущена у квітні 2018 року та підтримується за замовчування у Windows 10 v1803.

.NET Core є кросплатформеною реалізацією .NET Framework з відкритим кодом. Перша версія нової платформи була випущена в червні 2016 року компанією Microsoft. Платформа містить в собі середовище CoreCLR, що працює на будь-якій платформі. В CoreCLR входить компілятор RyuJIT. В серпні 2017 року відбувся реліз другої версії продукту. Вже анонсовано третю версію платформи. Вихід бета версії заплановано на кінець 2018 року.

Для реалізації алгоритму обрано .NET Core через те, що програма, написана на .NET Core є кросплатформеною та легкою. Для побудови програми потрібно лише завантажити компілятор Roslyn, що є у відкритому коді на GitHub за посиланням https://github.com/dotnet/roslyn.

Збірка – це одиниця виконання програмного коду, що складається з типів, інструкцій виконання, метаданих та ресурсів. Це звичайний файл з розширення .dll або .exe. Головна відмінність між цими двома типами – це те, що .exe – це виконуваний файл. Збірка має версію і всі типи в збірці мають ту ж саму версію.

Зазвичай всі типи належать одному простору імен і використовуються лише однією програмою. Але не є виключенням і такі збірки, що мають декілька просторів імен та виконуються декількома програмами.

В мові C# результатом компіляції є збірка. Збірки можуть мати посилання на інші збірки. Збірки компонується між собою спеціальною програмою компоновщиком (linker). Також в деякій літературі ця програма має назву «редактор зв’язків».

Простір імен – це така абстрактна множина, яка описує модель та збирає ідентифікатори в єдину групу. Ідентифікатори створенні в просторі імен асоціюються з цим простором. Це означає те, що однакові ідентифікатори можуть бути оголошенні в різних просторах, можуть мати однакові або різні значення. Простори імен вирішують проблему глобалізації даних. Різні мови програмування по різному підтримують простори імен або взагалі не підтримують їх.

Наприклад, мова C# та мова C++ мають однакову концепцію просторів імен – за допомогою оголошення блоку namespace. В подальшому простір імен можна імпортувати в будь-яку збірку. В мові Python це реалізовано за допомогою окремих модулів. В мові JavaScript відсутня концепція простору імен, але є багато механізмів для того щоб замінити цю підтримку.

C# - це мультипарадигмова мова програмування загального призначення, яка є сильно типізованою, об’єктно-орієнтованою, декларативною, функціональною і компонентною. Мова розроблена компанією Microsoft у 2000 році як мова розробки .NET додатків різного спрямування. На сьогоднішній день мова користується великою популярністю. Мова розроблена Андерсом Хейлсбергом. Остання версія мови 7.3 була випущена в 2018 році разом з останньою версією Visual Studio 2017 [15].

Мова розроблена базуючись на наступних концепціях:

- Ця мова є легкою у вивченні, сучасною та об’єктно-орієнтованою;
- Мова повинна надавати розробникам сильно типізацію даних, об’єктно-орієнтовану парадигму, реалізувати автоматичне очищення пам’яті;
- Програми написанні мовою C# повинні бути портативними та кросплатформеними;
- Програми написанні мовою C# повинні використовувати мінімальну кількість ресурсів.

Ця мова є дуже схожою на мови C та Java. Але не дивлячись на це мова C# отримала надзвичайно багато можливостей, яких і на сьогоднішній день немає у багатьох мовах програмування, наприклад, підтримка кортежів, перевантаження операторів, атрибути, шаблони, локальні функції, анонімні функції та типи, обробку подій і так далі.

Однією з основних технологій та фреймворків реалізованих на мові C# є технологія ASP.NET. ASP.NET – це популярний фреймворк, розроблений компанією Microsoft, створений для програмування веб-додатків, що підтримують протокол HTTP.

Вперше платформа була випущена в 2002 році разом з .NET Framework. На той момент платформа включала в собі засоби для розробки динамічних серверних сторінок Active Server Pager (ASP). В майбутньому платформа зазнала значних змін та на сьогоднішній день в її реалізацію входять наступні компоненти [18]:

- ASP.NET MVC – фреймворк для створення динамічних веб сторінок;
- ASP.NET WebAPI – фреймворк для побудови HTTP API;
- SignalR – фреймворк для створення додатків на базі веб-сокет технології;

Language Integrated Query (LINQ) – фреймворк, що дозволяє писати запити. Вперше було реалізовано в .NET Framework 3.5 і анонсовано як найкращу особливість цієї платформи. LINQ розширює бібліотеку додатковими запитами до колекцій, що дозволяє шукати, фільтрувати, конвертувати дані в будь-якій колекції.

До колекція які розширені бібліотекою LINQ відносяться:

- Масиви даних;
- Колекції даних;
- XML документи;
- Реляційні бази даних.

LINQ підтримує два стилі написання запитів: функціональний стиль та метод запитів. Функціональний метод реалізовано за допомогою

виклику ланцюгу функцій, а метод запитів – це реалізація запитів у вигляді SQL подібного синтаксису.

До стандартного публічного інтерфейсу бібліотеки відносять наступні методи:

- Select – проекція однієї колекції на іншу;
- Where – фільтрація вихідної колекції;
- SelectMany – проекція колекції колекцій на звичайну колекцію;
- Sum/Min/Max/Average – методи для отримання статитичних даних;
- Aggregate – метод для конвертації колекції в простий тип даних;
- Join/GroupJoin – метод, що виконує внутрішнє об’єднання колекцій;
- Take/TakeWhile – метод, що обирає перші n об’єктів колекції;
- Skip/SkipWhile – метод, що пропускає перші n об’єктів з колекції;
- First/FirstOrDefault – метод, що повертає перший елемент колекції (або елемент за замовчуванням якщо колекція пуста);
- Single/SingleOrDefault – метод, що повертає єдиний елемент колекції (або елемент за замовчуванням якщо колекція пуста);
- Any– метод що перевіряє чи задовольняє певній умові хоча б один елемент колекції;
- All – метод що перевіряє чи задовольняють певній умові всі елементи колекції;
- Count – метод, що повертає кількість елементів колекції;
- Contains – метод, що перевіряє на наявність певного елементу в колекції.

Також в LINQ API є певна кількість методів, що конвертує колекції в інший тип колекцій:

- AsEnumerable – повертає колекцію типу IEnumerable<T>;
- AsQueryable – повертає колекцію типу IQueryable<T>;
- ToArray – створює масив T[];
- ToList – створює список List<T>;

- ToDictionary – конвертує колекцію в словник Dictionary<K, T>;
- Cast/OfType – конвертує кожен елемент колекції.

Головною особливістю мови C# є перевантаження стандартних операторів. Це дозволяє розробнику розширити інтерфейс розроблених сутностей операціями додавання, віднімання, множення, приведення типу і так далі. Наступні оператори можуть бути перевантажені:

- Унарні оператори +, -, ++, --, ~, !;
- Бінарні оператори +, -, *, /, %, &, <<, >>;
- Оператори порівняння <, >, <=, >=, ==, !=;
- Умовні логічні оператори &&, ||;
- Оператор індексування колекції [];
- Оператор приведення типу T(K);
- Оператори присвоєння +=, -=, *=, /=.

```
public static Pollinator operator +(Pollinator first, Pollinator second)
{
   ThrowIf.NotEqual(first.Size, second.Size);

   var values = new double[first.Size];

   for (var i = 0; i < values.Length; i++)
   {
      values[i] = first._values[i] + second._values[i];
   }

   return new Pollinator(values);
}

public static Pollinator operator -(Pollinator first, Pollinator second)
{
   ThrowIf.NotEqual(first.Size, second.Size);

   var values = new double[first.Size];

   for (var i = 0; i < values.Length; i++)
   {
      values[i] = first._values[i] - second._values[i];
   }

   return new Pollinator(values);
}
```

Рисунок 3.1.2 – Приклад перевантаження операторів

В розробці алгоритму було використано певні патерни об’єктно-орієнтованого програмування, серед яких стратегія, фасад, інтерфейс, шаблонний метод, медіатор.

Патерн об’єктно-орієнтованого програмування – це загальний архітектурний метод вирішення проблем розробки програмного забезпечення. Патерни діляться на три великі групи [19]:

- Породжуючі патерни – призначенні для створенні нових сутностей класів;
- Структурні патерни – призначенні для зручної організації коду;
- Поведінкові патерни – призначені для зміни поведінки програми.

До групи породжуючих патернів відносять абстрактну фабрику, фабричний метод, будівельник, прототип, одиночка.

Абстрактна фабрика – це породжуючий патерн програмування, задачею якого є створення екземплярів класів одного інтерфейсу. Це один з найпопулярніших патернів у застосуванні. Абстрактна фабрика надає користувачу єдиний інтерфейс, зазвичай з одним методом «Створити», який повертає екземпляр класу, що відповідає певному інтерфейсу.

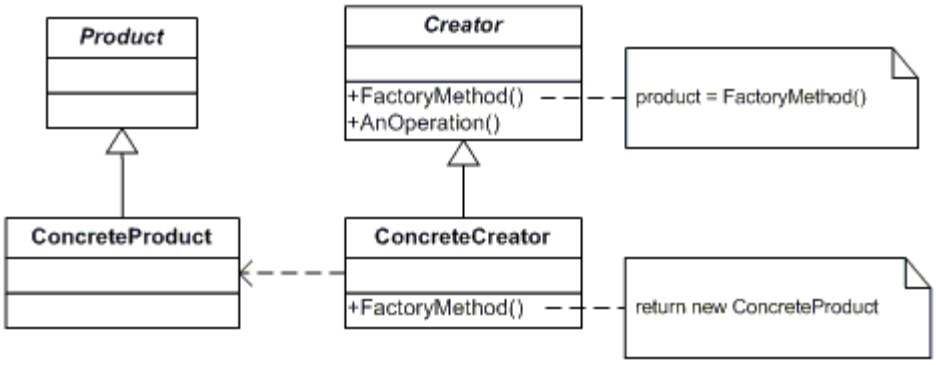

Рисунок 3.1.3 – UML діаграма абстрактної фабрики

Фабричний метод - це породжуючий патерн програмування, задачею якого є створення екземплярів класів, але інтерфейс фабрики дає змогу користувачу обирати який самий екземпляр класу йому потрібний. Відмінністю від абстрактної фабрики є те, що сам користувач визначає яка саме реалізація класу йому потрібна.

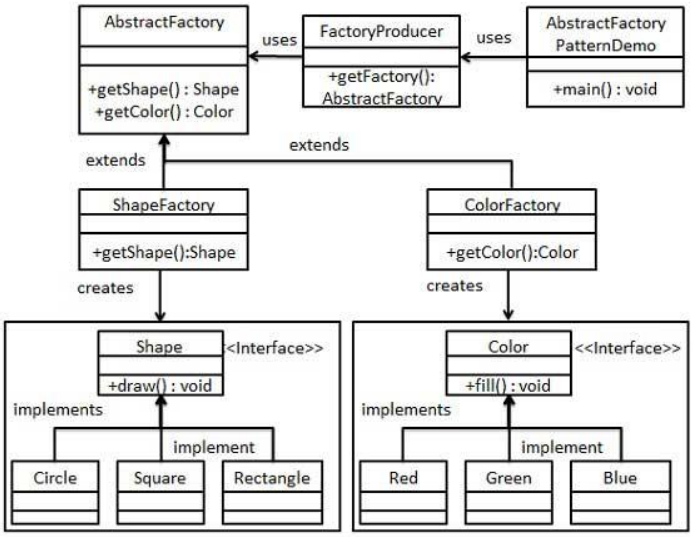

Рисунок 3.1.4 – UML діаграма фабричного методу.

Прототип – це породжуючий патерн, задачею якого є копіювання екземпляру класу. Прототип є розширенням базового функціоналу класу. Стандартна реалізація цього шаблону – єдиний метод, що повертає той самий клас, для якого він реалізований.

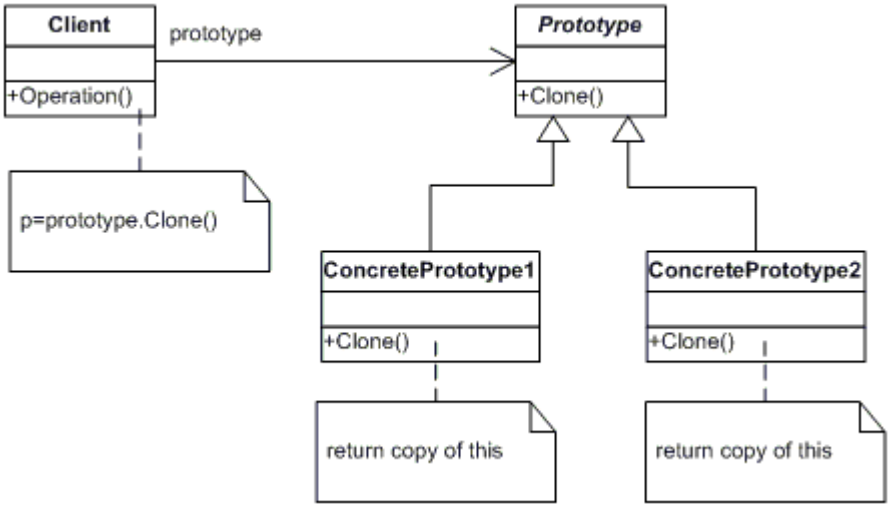

Рисунок 3.1.5 – UML діаграма прототипу

Наступна група патернів – структурні патерни. Серед цієї групи шаблонів найпопулярнішою є адаптер, декоратор, фасад.

Тестування програмного забезпечення – це процес контролю якості програмного забезпечення, ціллю якого є проведення групи тестів, які дають відповідь на питання чи задовольняє програма вимогам. Існує багато видів тестування, серед яких:

- Функціональне тестування;

- Модульне тестування;
- Інтеграційне тестування;
- Навантажувальне тестування;
- Тестування безпеки;
- Стресове тестування.

Створена програма була протестована за допомогою автоматичних модульних тестів та інтеграційних.

Тест – це процедура, що дозволяє перевірити працездатність розробленого програмного коду.

Модульне тестування або юніт тестування – це такий вид тестування, метою якого є перевірка коректності виконання окремих ізольованих модулів програми. Юніт тестування – це найпоширеніша практика у розробці комерційного програмного забезпечення. За його допомогою розробники перевіряють, що зміни певних частин коду не призвели до загальної зміни поведінки в вже відтестованих шматках програми. Чим більше покриття коду юніт тестами, тим менше можлива ймовірність помилки людини у виправленні або зміни вихідного коду. Існує багато бібліотек та фреймворків для покриття програми тестами. Наприклад, для мови C# це MSTest, XUnit, NUnit, Moq, TestDriven .NET, MbUnit.

При процесі юніт тестування модулів широко застосовується застосування fake- та mock-об’єктів. Mock-об’єкти – це такі екземпляри класів, що імітують інтерфейс певних класів. Вони застосовуються для того щоб ізольовано протестувати певний модуль без впливу поведінки та стану інших модулів. Mock-об’єкти описуються за допомогою методів-делегатів та створюються в процесі виконання тесту. Серед найпопулярніших бібліотек для створення fake-об’єктів існують наступні: Moq, jMock, EasyMock, NMock, PowerMock, sinon, Mockito.

Для тестування програми обрано бібліотеку xUnit. На сьогоднішній день ця бібліотека є найпопулярнішою серед аналогів та підтримується

компанією Microsoft. xUnit має ліцензію від Apache та має відкритий код. Спочатку бібліотека xUnit створювалась для мови програмування Smalltalk, але потім була модифікована та мігровано до мови C#, де отримала найбільшу популярність.

Для написання коду використовувалась загально поширена практика розробки програмного забезпечення через тестування Test Driven Development (TDD). Test Driven Development – це практика розробки програмного забезпечення, за якого спочатку створюється тест на певну поведінку сутності, а потім розширюється реалізація методу. За такого підходу розробник першим чином створює вимоги до наданого програмного інтерфейсу, а потім реалізує ці вимоги.

Поширювачем і розробником цієї практики вважається Кент Бек. Він казав, що методологія TDD заохочує розробників до написання такого коду, який легко підтримується. Також за допомогою тестів нові розробники можуть знайомитися з кодом програми, що покращує читабельність коду.

```
using Xunit;

public class Example
{
    [Fact]
    public void TestAreTrue()
    {
        Assert.IsTrue(true);
    }

    [Theory]
    [InlineData(2, 2, 4)]
    [InlineData(3, 3, 6)]
    public void TestAreCorrect(int a, int b, int expected)
    {
        // Arrange
        // Act
        var result = a + b;

        // Assert
        Assert.AreEqual(expected, result, "should be " + expected);
    }
}
```

Рисунок 3.1.6 – Приклад тестування на xUnit.

Цикл написання програми через тестування є наступним:

1. Створити тест – при розширенні вимог до програмного додатку, першим кроком є не написання цього функціоналу, а створення тесту на цю вимогу;
2. Перевірити результат тесту – в випадку якщо цей тест пройшов, то програма вже задовольняє новим вимогам;
3. Написати код – розширити базовий функціонал реалізацією нової вимоги;
4. Запустити всі тести – якщо хоча б один з тестів не пройшов, то розширена реалізація порушує іншу;
5. Почистити код.

Недоліками цієї методології є те, що певні задачі неможливо з самого початку покрити тестами, тому такі задачі неможливо вирішити спочатку написавши тест.

Інтеграційне тестування – це такий вид тестування, який перевіряє коректність зв'язків певних програмних модулів з іншими. Зазвичай цей вид тестування є наступним кроком тестування після модульного тестування.

Об'єктно-орієнтоване програмування (ООП) – це найпоширеніша методологія програмування, за якої програмне забезпечення розроблюється на базі опису класів та їх взаємодії. Основними поняттями ООП є наступні:

- Клас – структурна одиниця, що включає в себе стан (дані) та поведінку (методи);
- Об'єкт – екземпляр класу;
- Інкапсуляція – це такий стан, що дозволяє об'єднати методи та поля в рамках одного класу;
- Наслідування – це стан, що дозволяє описати новий клас на базі вже існуючого класу;
- Поліморфізм – це стан, що дозволяє один і той самий інтерфейс реалізовувати по різному;

- Абстракція даних – це виділення значної інформації з виключенням незначної.

В розробленій програмі використовується декілька широко поширених типів файлів, серед яких JSON, XML, CSV.

JavaScript Object Notation (JSON) – це текстовий формат даних, який базується на оголошенні об’єктів в мові JavaScript. Цей формат розроблений та впроваджений Дугласом Крокфордом – один з активних розробників JavaScript. Цей формат є дуже лаконічним. Через це набув широкого застосування, особливо в веб технологіях.

JSON формат представляє одну з двох структур:

1. Набір ключ-значення (об’єкти, словники, хеш-таблиці, записи, структури, асоціативний масив);
2. Впорядкований набір значень (масив, список, вектор, послідовність).

Значеннями можуть бути:

1. Об’єкти – це невпорядкована пара значень ключ : значення, що обмежене фігурними дужками «{}»;
2. Вектор – це впорядкована множина значень, що обмежена квадратними дужками «[]»;
3. Число;
4. Літерал – true/false/null;
5. Рядок – це впорядковані символи, що обмеженні подвійними лапками.

```
{
  "firstName": "Dima",
  "age": 22,
  "male": true,
  "address": {
    "city": "Kiev",
    "postalCode": "51613"
  },
  "responsibilities": ["developer", "student"]
}
```

Рисунок 3.1.7 – Приклад JSON об’єкту

JSON нотація і на сьогоднішній день широко розвивається. Остання версія була запропонована з появою нової версії JavaScript стандарту ECMAScript 5. Серед основних нововведень є підтримка коментарів, підтримка шістнадцяткової системи числення.

Альтернативою JSON формату є XML формат. Extensible Markup Language (XML) – це мова розмітки. Була запропонована у 1998 році і набула розвитку в мовах Extensible Hypertext Markup Language (XHTML), Rich Site Summary (RSS), Scalable Vector Graphic (SVG) і так далі.

Специфікація XML описує наступні структури:

5. Мову
6. Кодування
7. Обробку документів

Елементами XML документу є теги. Тег має наступну структуру - <Назва тегу>…</Назва тегу>. В випадку якщо тег пустий, то він має наступну структуру - </Назва тегу>. Тег може мати атрибути. Атрибути мають вигляд пари ключ-значення.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <address>
    <city>Kiev</city>
    <postalCode>51613</postalCode>
  </address>
  <age>22</age>
  <firstName>Dima</firstName>
  <male>true</male>
  <responsibilities>
    <element>developer</element>
    <element>student</element>
  </responsibilities>
</root>
```

Рисунок 3.1.8 – Приклад XML об’єкту

Comma Separated Values (CSV) – текстовий формат даних, що відповідає табличному представленню даних. Цей формат має найбільшу популярність серед розробників баз даних.

Наприклад, є таблиця наступного формату (Рис. 3.1.9).

1	**Data**	**Science**
Information	**A,b,c,d**	**next**

Рисунок 3.1.9 – Приклад таблиці для конвертації в CSV

Ця таблиця матиме наступний вигляд у CSV форматі (Рис. 3.1.10):

```
1,Data,Science
Information,”A,b,c,d,”,next
```

Рисунок 3.1.10 – Приклад контенту CSV файлу

Рефлексія – це одна з парадигм програмування, що означає процес в якому програма відслідковує та модифікує поведінку самої себе.

Рефлексія використовується для відслідковування стану та поведінки програми під час її виконання.

Рефлексійно-орієнтоване програмування складається з трьох частин:

- Самоперевірка;
- Самомодифікація;
- Самоклонування.

Перевагами використання рефлексії є наступні:

- Щоб отримати значення атрибуту в мові C# необхідно використовувати рефлексію;
- За допомогою рефлексії можна завантажувати збірку під час виконання програми;
- Це спосіб виконання приватних функцій та отримання значень приватних полів.

Нажаль рефлексія має дуже багато недоліків, серед яких основними є:

- Зниження швидкодії програми;
- Неможливість знаходження помилок під час компіляції;

- Дуже підвищується складність коду.

```
using System.Reflection

namespace ReflectionDemo {
   public class Foo
   {
      public void F()
      {
         // Some actions
      }
   }

   class Program
   {
      static void Main()
      {
         var foo = new Foo();

         // without reflection
         foo.F();

         // with reflection
         foo.GetMethod("F").Invoke(foo, null);
      }
   }
}
```

Рисунок 3.1.11 – Приклад використання рефлексії

Мікросервіс – це вид сервісно-орієнтованої архітектури, що спрямовано на взаємодію малих незалежних компонент між собою, що об’єднуються в єдиний сервіс. Мікросервіси отримали значне поширення за останнє десятиріччя в зв’язку з поширенням гнучких методологій розробки програмного забезпечення. У випадку веб технологій, мікросервіс – це тонкий веб-сервіс, що реалізує REST протокол обміну даних [20].

Розроблене програмне забезпечення було створене на базі мікросервісної архітектури, де мікросервіс – збірка, що надає публічні методи для взаємодії з нею. Наприклад, для заміни правил алгоритму необхідно лише змінити внутрішню реалізацію правил алгоритму, не змінюючи взаємодію між збірками. Саме тому алгоритм можна легко

модифікувати та змінювати його поведінку навіть під час виконання без компіляції всієї програми.

Вихідний код програми розміщено у приватному репозиторії в системі контролю версій Git. Git – це найпопулярніша система контролю версій вихідного коду. Створена розробником операційної системи Linux Лінусом Торвальдсом у квітні 2005 року. Система зберігає вихідний код та за допомогою системи можна отримати будь-яку попередню версію програми та порівняти її стан з початковим або кінцевим.

Під час реалізації алгоритму квіткового запилення спочатку було розроблено оригінальний алгоритм, після цього його модифіковано. Саме тому для порівняння роботи оригінального алгоритму і модифікованого необхідно було лише завантажити програмний код з системи контролю версій на момент реалізації оригінального алгоритму та порівняти його з модифікацією. Цю можливість надала система Git.

Applicable Programmable Interface (API) – це публічний інтерфейс програмного забезпечення, з яким користувач може взаємодіяти. API описує методи, константи, змінні, класи, типи та інші структури даних. Публічний програмний інтерфейс є необхідною частиною веб-сервісів, бібліотек, десктопних додатків.

У випадку веб-сервісів, програмний інтерфейс реалізовано через методи, що доступні через HTTP протокол. Якщо реалізований додаток є десктопним програмним забезпеченням, то його публічний інтерфейс доступний через публічні збірки цього додатку. Бібліотеки зазвичай мають документацію, де описанні методи та їх сигнатури, що є публічним програмним інтерфейсом цих бібліотек.

Applicable Programmable Interface є частиною інтеграції програмних додатків між собою. Кожна компонента цієї інтеграції виступає «чорним» ящиком, що надає декілька методів для публічної взаємодії. За таким принципом побудовано мікросервісну архітектуру – кожен мікросервіс реалізує дуже незначну частину функціоналу та надає тонке API іншим

сервісам. Користуючись публічним інтерфейсом кожної компоненти, сервіс змінює стан та поведінку кожної компоненти, виконуючи вимоги користувачів.

За рахунок API компонентами сервісу можуть бути додатки, написанні на різних мовах програмування та навіть різного спрямування.

Fluent Applicable Programmable Interface (Fluent API) – це одна з небагатьох методології розробки програмного забезпечення, головною ціллю якої є підвищення читабельності коду. Запропонований метод був Мартіном Фаулером.

Fluent API організується тим, що користувач викликає ланцюг методів одного об’єкта. За рахунок цього підвищується і інтуїтивність коду. На сьогодні розробники програмного забезпечення намагаються максимально слідувати концепції Fluent API.

Розроблене програмне забезпечення використовує декілька публічних бібліотек та фреймворків, що мають ліцензії. Серед них основними є NLog, Accord, CommandLineParser, Shouldly.

NLog – це бібліотека призначена для логування програмних подій. Бібліотека є гнучкою під будь-які вимоги користувача. Вона дозволяє логувати у консоль користувача, в файл та в базу даних. Остання версія бібліотеки вийшла у серпні 2018 року і отримала версію 4.5. Бібліотека має відкритий програмний код.

CommandLineParser – це бібліотека, що дозволяє конвертувати аргументи запуску у сутність певного класу. Бібліотека сумісна з платформою .NET Framework 4.0+, Mono 2.1+ Profile та .NET Core 1.0+. Вона не залежить від інших бібліотек, що робить її легкою та гнучкою у використанні (Рис. 3.1.12).

```
public class CommandLineArguments
  {
    [Option('g', "groups", Required = false, HelpText = "Groups count", Default = AlgorithmSettings.DefaultGroupsCount)]
    public int GroupsCount { get; set; }

    [Option('m', "ismin", Required = false, HelpText = "Is minimum", Default = false)]
    public bool IsMin { get; set; }

    [Option('n', "generation", Required = false, HelpText = "Number of generations", Default = AlgorithmSettings.DefaultMaxGeneration)]
    public int MaxGeneration { get; set; }
  }
```

Рисунок 3.1.12 – Приклад оголошення класу аргументів запуску

Shouldly – це бібліотека, що надає методи-розширення для перевірки відповідності значень. Найбільшу популярність бібліотека отримала у сфері тестування. Бібліотека реалізує FluentAPI, що робить код легко читабельним для його користувачів. У випадку не відповідності результату, бібліотека генерує програмне виключення, яке користувач може обробити або ігнорувати. Виключення має повідомлення, що пояснює чому програмне виключення відбулося. Бібліотека має реалізовані методи для порівняння простих структур даних, колекцій даних та структур даних, створених розробником.

```
p1.Equals(p2).ShouldBeTrue();
action.ShouldThrow<Exception>();
p.SequenceEqual(d).ShouldBeTrue();
```

Рисунок 3.1.13 – Приклад перевірок значень за допомогою Shouldly

Як було сказано вище, C# мова компілюється у код віртуальної машини, що називається проміжною мовою. Intermediate Language (IL) – це проміжна мова, у яку компілюється мова програмування C# для виконання її на спеціальній байт-машині. Це загальнодоступна мова, у яку може бути скомпільована будь-яка мова програмування, що підтримує .NET

Framework. На сьогоднішній день в проміжну мову компілюються і програми, що підтримують Mono та .NET Core.

Насправді код є дуже схожим на інструкції мови C#, але насправді ці дві мови мають багато відмінностей.

```
internal Task \u003CPolinateOnceAsync\u003Eb__1(Task<Pollinator[]> task)
{

AlgorithmPerformer.\u003C\u003Ec__DisplayClass12_0.\u003C\u003CPolinateOnceAsync\u
003Eb__1\u003Ed stateMachine = new
AlgorithmPerformer.\u003C\u003Ec__DisplayClass12_0.\u003C\u003CPolinateOnceAsync\u
003Eb__1\u003Ed();
    stateMachine.\u003C\u003E4__this = this;
    stateMachine.task = task;
    stateMachine.\u003C\u003Et__builder = AsyncTaskMethodBuilder.Create();
    stateMachine.\u003C\u003E1__state = -1;

stateMachine.\u003C\u003Et__builder.Start<AlgorithmPerformer.\u003C\u003Ec__Display
Class12_0.\u003C\u003CPolinateOnceAsync\u003Eb__1\u003Ed>(ref stateMachine);
    return stateMachine.\u003C\u003Et__builder.Task;
}
```

Рисунок 3.1.14 – Приклад проміжної мови для асинхронної машини стану

Найбільш складними для компіляції у проміжну мову є асинхронні оператори, що є часто вживаними у сучасному стандарті мови C#.

Асинхронні задачі в мові C# виконуються у окремому потоці, шо виділяється загальним пулом платформи .NET Framework. Тобто у мові C# асинхронні задачі насправді є паралельними задачами, що виконуються незалежно від головного потоку.

Різниця між асинхронними задачами та звичайними задачами полягає у тому, що асинхронні задачі завжди повертають значення класу Task або Task<T>. Також асинхронні задачі мають в сигнатурі спеціальне слово async. Також у окремих випадках асинхронні задачі можуть повертати значення типу void та ValueTask<T>.

Асинхронний метод так само як і звичайний синхронний метод може приймати будь-яку кількість аргументів, може бути методом класу або методом делегатом. Єдина відмінність – це використання ключових слів async/await.

```
    public Task<Pollinator> ApplyRuleAsync(Pollinator pollinator,
GlobalPollinationRuleArgument ruleArgument)
                                {
                                                var bestPollinator =
ruleArgument.BestPollinator;
                                                return Task.Run(() =>
                                                {
                                                                var distanceDifference =
(pollinator - bestPollinator)
                .CountFunction(FunctionStrategies.FunctionStrategies.DistanceFun
ction);
                                                                var lambda =
FunctionStrategies.FunctionStrategies.LambdaFunction(distanceDifference);
                                                                var rand =
FunctionStrategies.FunctionStrategies.MantegnaFunction(lambda);
                                                                var result = pollinator +
rand * (pollinator - bestPollinator);
                                                                return result;
                                                });
                                }
```

Рисунок 3.1.15 – Приклад асинхронного методу

Програмний репозиторій налаштований таким чином, щоб завантажити вихідний код на сервер можна було тільки у випадку, якщо проект компілюється без помилок та якщо кожен тест проходить з успішним результатом. Для цього використовується автоматична система компіляції проектів Travis CI. Travis CI – це веб сервіс, що

використовується як засіб для побудови проекту та подальшої доставки її користувачу, розроблений у 2012 році.

В цілях дипломного проекту ця система використовувалась для автоматичної перевірки працездатності програмного забезпечення. Система підтримує багато мов програмування, серед яких C# (.NET Core 1.0+), C++, Python, Ruby On Rails, Java, JavaScript та багато інших. Програмний код сервіс є у відкритому доступі в GitHub репозиторії.

```
language: csharp
solution: LevyFlightSharp.sln
mono: none
dotnet: 2.1
install:
 - dotnet restore LevyFlightSharp.sln
script:
 - dotnet build
 - dotnet test "src/LevyFlight.Tests/LevyFlight.Tests.csproj" --filter
Category=Unit
branches:
 only:
  - gh-pages
  - /.*/
```

Рисунок 3.1.16 – Приклад налаштування Travis CI

3.2. Опис основної програми

Для аналізу результатів алгоритму багатокритеріальної оптимізації створено програмне забезпечення, що дозволяє виконувати модифікований алгоритм квіткового запилення для рішення задачі багатокритеріальної оптимізації.

Програма написана мовою C# і використовує .NET Core 2.1. Сама програма складається з 11 збірок:

- LevyFlight.Api – збірка, що містить публічний інтерфейс алгоритму;

- LevyFlight.Common – збірка, де знаходиться група загальних методів, що не відносяться до алгоритму (перевірки на коректність значень, генератор випадкових чисел і так далі);
- LevyFlight.Domain – збірка, що містить опис алгоритму та його моделей даних;
- LevyFlight.Entities – збірка, що містить опис домених моделей та деяких розширень.
- LevyFlight.Examples – збірка, що містить опис benchmark функцій;
- LevyFlight.Extension – збірка, що містить розширення базового функціоналу класів;
- LevyFlight.Logging – збірка, що містить сутності необхідні для логування;
- LevyFlight.Logic – збірка, що містить класи-фабрики;
- LevyFlight.Startup – збірка, що є вхідною точкою консольної програми;
- LevyFlight.TestHelper – збірка, що містить сутності необхідні для тестування;
- LevyFlight.Tests – збірка, що містить модульні та інтеграційні тести.

Режим роботи та вхідні параметри алгоритму задаються як параметри при запуску консольного додатку. Всі параметри є не обов’язковими та мають значення за замовченням. Вхідними параметрами програми є:

- Кількість груп запилювачів (параметр «-g» або «--groups») – значення за замовченням 2;
- Пошук мінімуму або максимуму (параметр «-m» або «--ismin») – значення за замовченням 1;
- Кількість ітерацій алгоритму (параметр «-n» або «--generation») – значення за замовченням 30;

- Ймовірність переключення між локальним і глобальним запиленням (параметр «-p» або «--probability») – значення за замовченням 0.91;
- Кількість запилювачів (параметр «-c» або «--pollinators») – значення за замовченням 25;
- Ймовірність скидання (параметр «-r» або «--reset») – значення за замовченням 0.01;
- Кількість змінних функції оптимізації (параметр «-v» або «--variables») – значення за замовченням 30;
- Назва функції оптимізації (параметр «-f» або «--function») – значення за замовченням «Z1»;

Серед функцій, що містить збірка LevyFlight.Examples є наступні benchmark функції для перевірки коректності роботи алгоритму багатокритеріальної оптимізації:

$$Z1\colon f_1(x) = x_1; f_2(x) = g\left(1 - \frac{f_1}{g}\right)^2; g = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \quad (1)$$

$$Z2\colon f_1(x) = x_1; f_2(x) = g\left(1 - \sqrt{\frac{f_1}{g}}\right); g = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \quad (2)$$

$$Z3\colon f_1(x) = x_1; f_2(x) = g\left(1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g}\sin(10\pi f_1)\right); g = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \quad (3)$$

Для того щоб перевірити роботу алгоритму з будь-якою з цих трьох функцій, треба запустити програму, вказавши назву функції «Z1», «Z2» або «Z3». Також можлива перевірка роботи алгоритму однокритеріальної оптимізації. Серед benchmark функцій однокритеріальної оптимізації визначені наступні:

- AckleyFunction

$$f(x) = -20\exp(-\frac{1}{5} * \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)) + 20 + e.$$

- GriewankFunction

$$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1.$$

- RastriginFunction

$$f(x) = A_n + \sum_{i=1}^{n}[x_i^2 - Acos(2\pi x_i)].$$

- RosenbrockFunction

$$f(x) = \sum_{i}^{n-1}[(x_i - 1)^2 + 100(x_{i+1} - x_i^2)^2].$$

- SphereFunction

$$f(x) = \sum_{i=1}^{n} x^2.$$

Програма написана таким чином, щоб скомпілювавши її можна було отримати готову бібліотеку, яку можна підключити до будь-якого іншого додатку. Для цього створено збірку LevyFlight.Api, яка надає публічний програмний інтерфейс для користування алгоритмом. Збірка містить лише три класи: Extremum, MaxExtremum, MinExtremum. Клас Extremum є базовим класом та визначає публічний інтерфейс користувача. Він містить конструктор, два методи та одну подію.

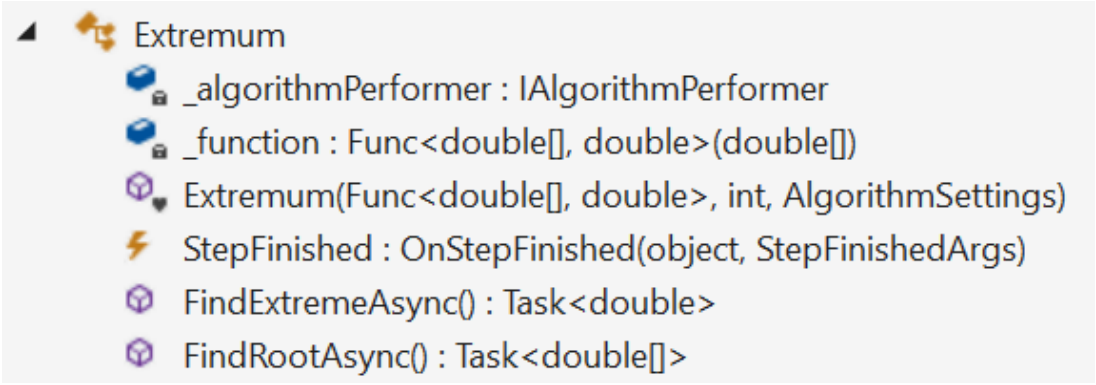

Рисунок 3.2.1 - Інтерфейс класу Extremum

При створені клас приймає функцію оптимізації, кількість змінних функції та об’єкт, що описує параметри алгоритму. В класі реалізовано два методи FindExtremeAsync та FindRootAsync, що повертають результат

роботи алгоритму у вигляді рішень або оптимального значення функції. Також клас містить одну подію StepFinished, на яку можна підписатися.

Програма за допомогою інструментів логування може записувати результат покрокового виконання у файл. Назва цього фалу має наступний формат: [рік]-[місяць]-[день].log. Наприклад «2018-01-02.log». В залежності від конфігурації збірки LevyFlight.Logging програма записує в файл кінцевий результат або результат виконання кожного кроку. Той самий результат може бути записаний і в консоль користувача.

Конфігурацією збірки LevyFlight.Logging є файл NLog.config. Це документ, що має чітко визначену XML структуру. Він містить дві основні секції:

- Target – місце куди буде виконуватись логування програми. Об’єктами логування можуть бути консоль, файл, e-mail і так далі.
- Rules – правила логування для об’єктів. Визначають які типи повідомлень будуть виведені в об’єкт логування.

Існують різні рівні логування, що визначають типи повідомлень. Наступні рівні логування доступні в програмі:

- Trace – найнижчий рівень логування, необхідний для розробників;
- Debug – рівень логування, що використовується для тестування;
- Info – рівень логування, що містить повідомлення інформаційного характеру;
- Warn – рівень логування необхідний для попередження користувачів/розробників;
- Error – рівень логування, що визначає помилки програми;
- Fatal – найвищий рівень логування, що визначає аварії програми.

Програма визначає лише всі рівні логування, окрім рівня Trace. На рівні Debug користувач отримує повідомлення покрокового виконання програми. На рівні Info користувач отримує лише кінцевий результат виконання алгоритму. На рівні Warn розробник може отримати

попередження при некоректній поведінці програми. На рівні Error та Fatal логуються помилки виконання програми (Рис. 3.2.2).

2018-11-04 20:24:39.8514 DEBUG Start step 1

2018-11-04 20:24:39.9809 DEBUG Best pollinator after step 1 is -9,84949705649038;-5,68502512450507;-0,0491894319346441;-0,655096822087428;9,70494036143132;0,93183613733521;-9,85600119355993;0,779753299677956;4,56734214952974;-0,189235355525735;-5,88460678526059;5,17931415880874;8,26295307440445;-4,22610532983259;-12,6705477745882;-0,137946614524514;-1,4693084921943;11,3188756827652;-11,1589877620884;-6,26715783359916;1,30219379911524;15,4835149327988;-13,7550025290514;-0,749804351305124;5,64337791397488;-4,01232072393579;-5,48122851773199;-9,47936207149387;6,96441831421601;16,2454479882321

2018-11-04 20:24:42.3751 ERROR Pollinator got NaN or +/- Infinity. Pollinator -∞;∞;∞;∞;-∞;2,14714587276305E+301;∞;-∞;-∞;∞;∞;-∞;-∞;∞;∞;∞;∞;-∞;∞;∞;-∞;-∞;∞;∞;∞;∞;∞;∞;∞;-∞;-∞

Рисунок 3.2.2 – Приклад повідомлень різного рівня

Параметри алгоритму визначає клас AlgorithmSettings. Він містить декілька обов'язкових полів та конструктор, що ініціалізує ці поля. Також цей клас визначає параметри алгоритму за замовченням.

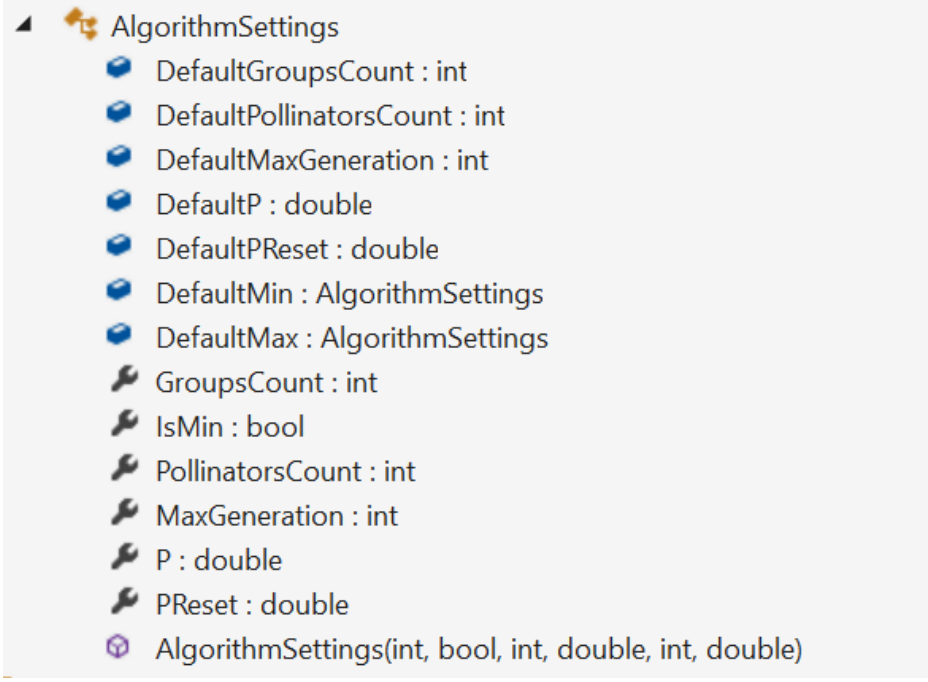

Рисунок 3.2.3 – Інтерфейс класу AlgorithmSettings

Це один з об'єктів, що є необхідним для правильної роботи алгоритму. Також він визначає поведінку та стан алгоритму, а саме

переключення між локальним та глобальним запиленням, ймовірність скидання рішення, кількість груп запилювачів та кількість запилювачів.

Так як алгоритму квіткового запилення призначено для виконання однокритеріальної оптимізації, створено клас, що за допомогою функції скаляризації зводить декілька функцій оптимізації в одну. Це клас MultifunctionStrategy. Він приймає на вхід декілька функцій оптимізації та повертає скаляризовану функцію.

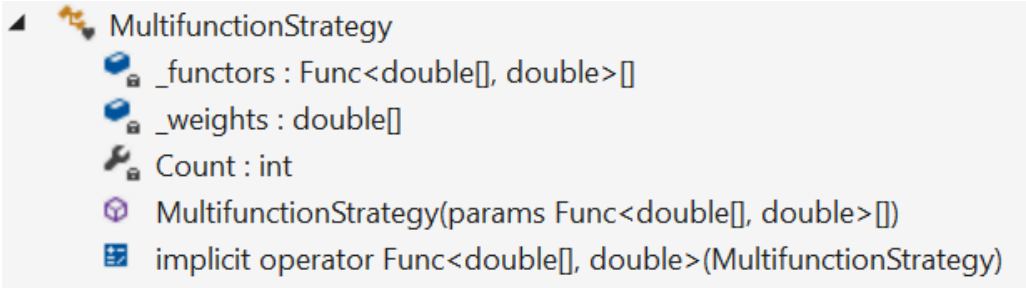

Рисунок 3.2.4 – Інтерфейс класу MultifunctionStrategy

Клас використовує адитивний метод скаляризації, тобто за допомогою операції додавання зводить декілька критеріїв до одного.

Збірка LevyFlight.Domain є ядром всього додатку та однозначно визначає домені моделі алгоритму. Саме ця збірка містить опис алгоритму оптимізації, опис моделей та правил алгоритму. Один з основних класів - це AlgorithmPerformer. В цьому класі описано код алгоритму квіткового запилення. Він має наступні методи:

- PollinateAsync – асинхронно виконує алгоритм та повертає «найкращого» запилювача;
- PolinateOnceAsync – асинхронно виконує одну ітерацію алгоритму та повертає «найкращого» запилювача;
- GlobalPollinationAsync – асинхронно виконує глобальне запилення та повертає нового запилювача;
- LocalPollinationAsync – асинхронно виконує локальне запилення та повертає нового запилювача;
- PostOperationAsync – асинхронний метод, що приймає рішення про заміну запилювачів (в випадку покращення рішення).

Клас має конструктор, що приймає параметри алгоритму, кількість змінних функції оптимізації, функцію оптимізації, фабрику запилювачів, правила що описують гілки алгоритму, клас для логування. Сам клас не описує гілки алгоритму, він лише приймає правила, що їх описують. Тобто для зміни поведінки алгоритму необхідно редагувати правила алгоритму, а не клас, що описує алгоритм.

Рисунок 3.2.5 – Інтерфейс класу AlgorithmPerformer

Правила, що описують алгоритм, описані єдиним шаблонним інтерфейсом IRule. Цей клас має єдиний метод ApplyRuleAsync, що приймає на вхід запилювача та додаткові параметри, необхідні для правильної роботи гілки алгоритму (випадковий запилювач, найкращий запилювач). Метод повертає нового запилювача.

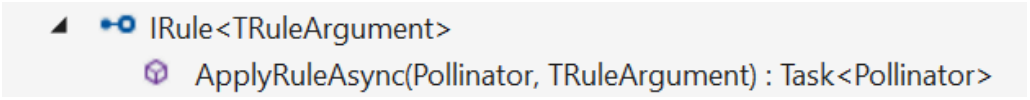

Рисунок 3.2.6 – Інтерфейс IRule

Цей інтерфейс реалізують три сутності:

- GlobalPollinationRule;
- LocalPollinationRule;
- ResetPollinationRule.

Кожна з цих сутностей визначає модель даних, що є входом для правил алгоритму. Наприклад, для глобального запилення ця сутність містить найкращого запилювача.

Збірка ще визначає інтерфейс фабрики, що створює екземпляри класів типу IAlgorithmPerformer. Це єдиний клас, який доступний публічно для використання за межами збірки.

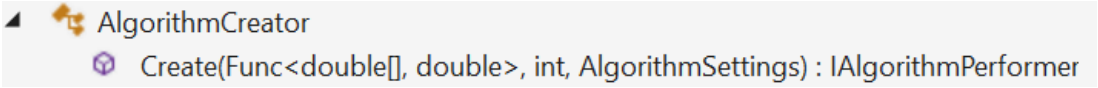

Рисунок 3.2.7 - Інтерфейс класу AlgorithmPerformer

Розроблена програма є консольним додатком, який компілюється з точкою входу в проекті LevyFlight.Startup. Ця збірка має два перевантажених методи Main та опис консольних аргументів запуску.

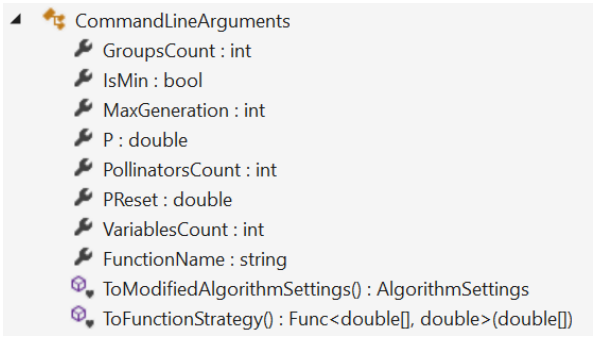

Рисунок 3.2.8 – Інтерфейс класу CommandLineArguments

Однією з основних збірок проекту є збірка LevyFlight.Entities, в якій описано моделі алгоритму – запилювачі та групи запилювачів. Запилювач – це вектор рішень, а група запилювачів – колекція запилювачів.

Запилювач має реалізовані перевантаженні оператори додавання, віднімання, множення і ділення. Додавати та віднімати можна двох запилювачів (додавання векторів), а множити і ділити запилювач можна і на скаляр.

Також запилювач і група запилювачів реалізують патерн відвідувач, що дає змогу легко розширювати їх функціонал (Рис. 3.2.9).

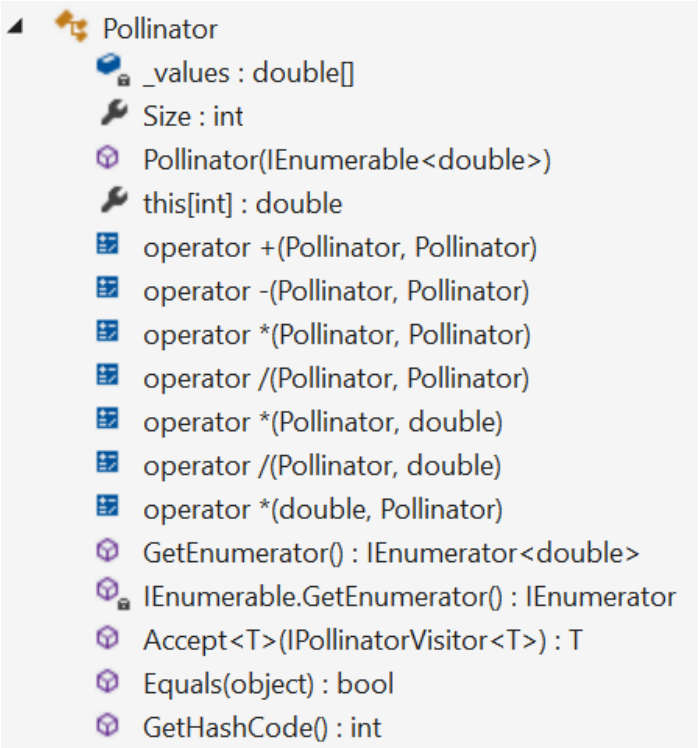

Рисунок 3.2.9 – Інтерфейс класу Pollinator

Програмне забезпечення може бути скомпільоване в вигляді бібліотеки або в вигляді консольного додатку. Для побудови бібліотеки проект необхідно будувати зі стартовою збіркою LevyFlight.Api, а для побудови консольного додатку необхідно використати початкову збірку LevyFlight.Startup.

В випадку компіляції вихідного коду у вигляді бібліотеки, користувач отримає збірку LevyFlight.Api.dll, яку він зможе підключити у будь-який додаток та користуватися алгоритмом.

Для того щоб вказати параметри алгоритму, треба викликати програму вказавши аргументи запуску. Вони є необов’язковими і в разі відсутності хоча б одного з аргументів, програма встановить їх значення за замовченням.

```
LevyFlight.Startup.exe -f griewank -m -n 1000
```

Рисунок 3.2.10 – Приклад запуску консольного додатку

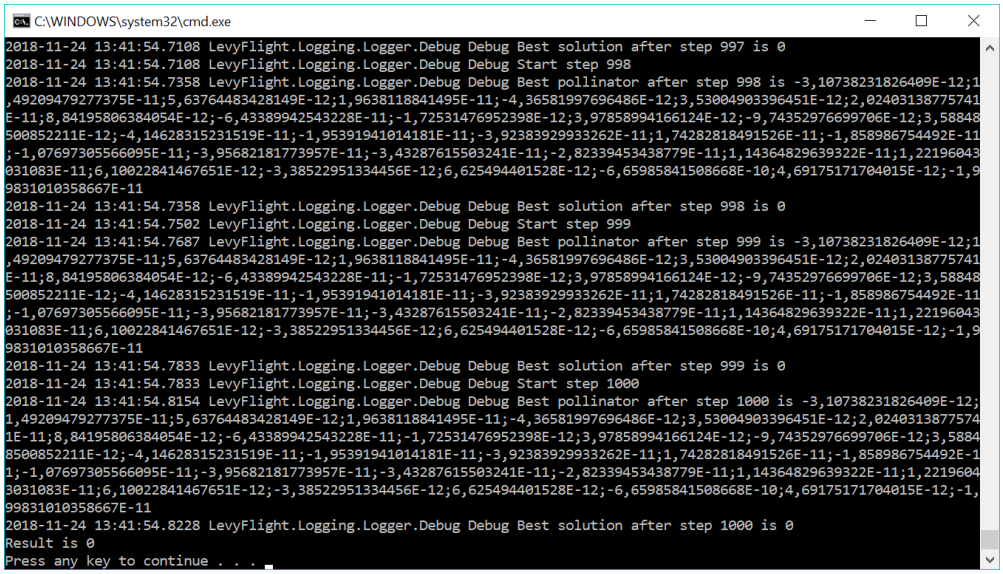

Рисунок 3.2.11 – Консольний інтерфейс програми

Збірка LevyFlight.Tests включає в себе автоматичні тести для часткової перевірки працездатності алгоритму. Вся програма покрита двома видами тестів:

- Модульні тести – перевіряють працездатність окремих класів без взаємодії з іншими, наприклад, тести на коректність роботи класу «запилювач»;
- Інтеграційні тести – перевіряють працездатність всього алгоритму з реальними сутностями;

Інтеграційні тести запускають алгоритм з певними налаштуваннями 500 разів та перевіряють що абсолютне відхилення відомого фронту Парето (або глобального мінімуму в випадку однокритеріальної оптимізації) не перевищую задану похибку. Інтеграційні тести повністю тестують працездатність зовнішнього публічного інтерфейсу, що надається користувачу.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Визначення оптимальних параметрів алгоритму

Було проведено серію випробовувань модифікованого алгоритму з різними початковими параметрами. Модифікований алгоритм запускався по 30 разів на кожній з функцій.

Оптимальними в результаті випробовувань вважалися ті параметри, за яких алгоритм знаходить оптимальний розв'язок з найбільшою точністю за час, що не перевищує хвилину.

Точність знайденого розв'язку аналізувалась за наступною формулою:

$$E_f = \sum_{i=1}^{N} (F_i^T - F_i^a)^2 \quad (1)$$

Де F_i^T – відоме значення фронту Парето, F_i^a – значення отримане за допомогою розрахунків.

Наступні методи скаляризації були протестовані:

1. Адитивний метод;
2. Мультиплікативний метод;
3. Канонічний адитивний-мультиплікативний метод;

Цільові функції оптимізації [21]:

$$F1\colon f_1(x) = x_1; f_2(x) = g\left(1 - \frac{f_1}{g}\right)^2; g = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \quad (1)$$

$$F2\colon f_1(x) = x_1; f_2(x) = g\left(1 - \sqrt{\frac{f_1}{g}}\right); g = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \quad (2)$$

$$F3\colon f_1(x) = x_1; f_2(x) = g\left(1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g}\sin(10\pi f_1)\right); \quad (3)$$

$$g = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1}$$

Випробовування проводились на машині з наступними характеристиками:

- 64 розрядний двоядерний процесор з частотою 2.7 ГГц
- 8 Гб оперативної пам’яті

В таблицях нижче наведено середній результат по всім запускам модифікованого алгоритму.

1. Параметри алгоритму: кількість запилювачів в групі - 5, кількість груп - 5, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – адитивний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	5.26e-9	7.57e-17
F2	6.20e-11	8.37e-17
F3	4.82e-9	8.16e-15

Табл. 4.1.1. Тестування роботи модифікованого алгоритму №1

2. Параметри алгоритму: кількість запилювачів в групі - 5, кількість груп - 5, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – мультиплікативний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	2.71e-11	2.90e-20
F2	8.01e-12	1.90e-20
F3	2.33e-11	2.97e-20

Табл. 4.1.2. Тестування роботи модифікованого алгоритму №2

3. Параметри алгоритму: кількість запилювачів в групі - 5, кількість груп - 5, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – канонічний адитивний-мультиплікативний метод.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	7.21e-8	7.48e-13
F2	6.47e-8	7.77e-13
F3	1.28e-8	9.98e-14

Табл. 4.1.3. Тестування роботи модифікованого алгоритму №3

4. Параметри алгоритму: кількість запилювачів в групі - 2, кількість груп - 12, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – адитивний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	5.61e-10	3.12e-17
F2	8.34e-11	3.31e-17
F3	2.81e-10	2.50e-16

Табл. 4.1.4. Тестування роботи модифікованого алгоритму №4

5. Параметри алгоритму: кількість запилювачів в групі - 2, кількість груп - 12, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – мультиплікативний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	2.53e-11	3.25e-22
F2	1.53e-12	7.77e-21
F3	1.03e-11	4.13e-21

Табл. 4.1.5. Тестування роботи модифікованого алгоритму №5

6. Параметри алгоритму: кількість запилювачів в групі - 2, кількість груп - 12, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – канонічний адитивний-мультиплікативний метод.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	8.12e-8	1.67e-14
F2	7.55e-8	2.24e-13
F3	6.37e-7	5.05e-15

Табл. 4.1.6. Тестування роботи модифікованого алгоритму №3

7. Параметри алгоритму: кількість запилювачів в групі - 5, кількість груп - 10, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – адитивний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	3.77e-9	4.11e-18
F2	3.82e-11	1.75e-18
F3	7.31e-10	1.51e-16

Табл. 4.1.7. Тестування роботи модифікованого алгоритму №7

8. Параметри алгоритму: кількість запилювачів в групі - 5, кількість груп - 10, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – мультиплікативний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	2.36e-12	4.87e-21
F2	1.53e-12	8.17e-22
F3	1.14e-12	7.91e-21

Табл. 4.1.8. Тестування роботи модифікованого алгоритму №8

9. Параметри алгоритму: кількість запилювачів в групі - 5, кількість груп - 10, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – канонічний адитивний-мультиплікативний метод.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	1.09e-9	6.62e-14
F2	1.15e-8	2.68e-15
F3	9.98e-9	9.29e-15

Табл. 4.1.9. Тестування роботи модифікованого алгоритму №9

10. Параметри алгоритму: кількість запилювачів в групі - 10, кількість груп - 5, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – адитивний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	8.89e-9	6.37e-17
F2	1.02e-10	9.85e-17
F3	3.11e-9	2.36e-15

Табл. 4.1.10. Тестування роботи модифікованого алгоритму №10

11. Параметри алгоритму: кількість запилювачів в групі - 10, кількість груп - 5, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – мультиплікативний.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	1.13e-11	9.01e-21
F2	4.11e-12	9.22e-23
F3	9.12e-13	1.22e-20

Табл. 4.1.11. Тестування роботи модифікованого алгоритму №11

12. Параметри алгоритму: кількість запилювачів в групі - 10, кількість груп - 5, імовірність глобального запилення $p = 0.91$, кількість ітерацій – 1000, метод скаляризації – канонічний адитивний-мультиплікативний метод.

Функція	E_f**, 1000 ітерацій**	E_f**, 5000 ітерацій**
F1	3.91e-9	3.30e-13
F2	1.14e-8	7.20e-14
F3	1.34e-8	8.85e-14

Табл. 4.1.12. Тестування роботи модифікованого алгоритму №12

Як видно з результатів роботи, найкращим методом скаляризації був мультиплікативний метод, а поєднання адитивного методу з мультиплікативним погіршило точність знайденого розв'язку. Також з результатів видно, що збільшення кількості груп покращує результат більше, ніж збільшення кількості запилювачів в групі. Тому для порівняння оригінального алгоритму і модифікованого було прийнято рішення взяти кількість груп, що дорівнює 10, а кількість запилювачів в групі 5. За цих параметрів алгоритм дуже швидко знаходить рішення з малим відхиленням від відомого фронту Парето.

4.2. Порівняльний аналіз

На основі тестування, було встановлено, що оптимальними параметрами алгоритму є наступні:

- Кількість запилювачів в групі – 5;
- Кількість груп – 10;
- Імовірність глобального запилення $p = 0.85$;
- $\lambda \in [0.3; 1.99]$;
- Кількість ітерацій – 1000;
- Метод скаляризації - мультиплікативний

Комп’ютерне тестування запропонованого алгоритму виконувалося на стандартних benchmark функціях. Модифікований і оригінальний алгоритм запускалися по 30 разів на кожній з функцій. В таблиці (Табл 4.2.1) наведені результати тестування.

Функція	E_f**, 1000 ітерацій**		E_f**, 5000 ітерацій**	
	MFPA	FPA	MFPA	FPA
F1	3.91e-9	9.85e-5	3.30e-13	1.53e-9
F2	1.14e-8	3.40e-5	7.20e-14	7.63e-9
F3	1.34e-8	1.14e-5	8.85e-14	1.41e-8

Табл. 4.2.1. Порівняльний аналіз модифікованого алгоритму квіткового запилення (MFPA) і оригінального (FPA).

Як видно з результатів випробовувань, модифікований алгоритм у всіх випадках знаходить розв’язок ближче до фронту Парето, ніж оригінальний алгоритм.

5. РЕЗУЛЬТАТИ НАУКОВО-ДОСЛІДНОЇ ПРАКТИКИ

Під час науково-дослідної практики було розроблено сайт, що використовував математичну бібліотеку. Бібліотека була розроблена тією ж командою, шо і розробляла сайт. Сайт працює по HTTPS протоколу та має публічний програмний інтерфейс у вигляді 2 методів:

- Метод POST за адресою api/func – метод, що знаходить значення вказаної функції у точках, що передані в тілі запиту. Функція описана у тілі запиту;
- Метод GET api/func/last – метод, що повертає результат останнього обчислення.

Сайт було розроблено у вигляді мікросервісу, що використовується іншим зовнішнім сервісом. Складність полягала у тому, що бібліотека обчислення повинна була створити функцію на базі запиту та обчислити її значення у вказаних точках. Саме ця бібліотека і розроблювалась протягом науково-дослідної практики.

Бібліотека була розроблена та підключена до веб-сервісу. Вона мала простий публічний інтерфейс, який веб-сервіс і використовував. Публічний інтерфейс було попередньо сплановано для того, щоб веб-сервіс та математична бібліотека могли розроблятися незалежно один від одного.

ВИСНОВКИ

Модифікований алгоритм квіткового запилення є сучасною метаевристикою, яка здатна розв'язувати проблему багатокритеріальної оптимізації без будь-якої спеціальної інформації про область визначення, крім тої, яка необхідна для обчислення цільової функції.

За допомогою методу скаляризації алгоритм приведено до форми, що дозволяє розв'язувати задачу багатокритеріальної оптимізації.

Тестування на відомих benchmark функціях продемонструвало, що запропонована модифікація методу є ефективною при розв'язанні проблеми багатокритеріальної оптимізації у випадку сильно нелінійних задач зі складними обмеженнями й різноманітними типами оптимальних за Парето множин.

Також тестування показало те, що для даного методу найкраще застосовувати метод мультиплікативної скаляризації. Саме при застосуванні цього методу отриманий результат мав найменше відхилення від відомого фронту Парето. Результати показали, що метод який об'єднує два методи скаляризації виявився найгіршим в сенсі швидкості пошуку та точності отриманих результатів.

Так як розроблена програма може бути застосована як бібліотека, то розроблений метод вже можна використовувати як зовнішню залежність в будь-якому програмному забезпеченні. Розроблена збірка має простий публічний програмний інтерфейс та її можна легко налаштувати. Саме тому даний метод може легко поширюватись через будь-яку систему управління зовнішніми пакетами, наприклад, NuGet для платформ Microsoft.

Для майбутнього покращення методу можна спробувати поєднати метод з іншими відомими евристичними методами, наприклад, додати до запилювачів елемент «привабливості» як в методі світлячків та в залежності від цієї характеристики змінити поведінку методу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. M. Blaug, “Economic Theory in Retrospect”, Economic London School, London, Great Britain, 1962. – P. 540-561 – XVII, 627 P. – ISBN 5-86461-151-4.
2. A. Osyzka, “Multicritaria optimization for engineering design”, American Society of Civil Engineers, Virginia, 1996
3. С.Е. Ануфрієнко, «Метод Парето рішення багатокритеріальних задач», Ярославський національний університет, Ярославль, Росія, 2004. – с. 6-8 – 24 с.
4. L. X. Li, Z. J. Shao and J. X. Qian, “An optimizing Method based on Autonomous Animals: Fish Swarm Algoritm”, System Engineering Theory & Practice, 2002.
5. X. S. Yang, “Nature-Inspired Metaheuristic Algorithms”, Luniver Press, 2008.
6. S. L. Ukasik and AK. SÃlawomirZ, “Firefly algorithm for Continuous Constrained Optimization Tasks”, 1st International Confernce on Computational Collective Intelligence, Semantic Web, Social Networks and Multiagent Systems, Springer-Verlag Berlin, Heidelberg 2009. – p.169-178.
7. X. S. Yang, “A New Metaheuristic Bat-Inspired Algorithm”, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK, 2010.
8. K. E. Parsopoulos and M. N. Vrahatis, “Particle Swarm Optimization and Intelligence”, Information Science Reference, Hershey, New-York, 2010
9. T. P. Gill, “The Doppler Effect. An Introduction To The Theory of the Effect”, Logos Press, Academic Press, 1965.
10. X. S. Yang, Flower Pollination Algorithm for Global Optimization. Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK, 2013.

11. N. Diab, “Recent Advances in Flower Pollination Algorithm”, International Journal of Computer Applications Technology and Research Volume 5– Issue 6, 2016. – P. 338 – 346 - ISSN:-2319–8656.
12. S. Lukasik and P. A. Kowalski, “Study of Flower Pollination Algorithm for Continuous Optimization”, Department of Automatic Control and Information Technology, Cracow University of Technology, 2016
13. Y. Zhou, R. Wang and Q. Luo, “Elite opposition-based flower pollination algorithm”, College of Information Science and Engineering, Guangxi University for Nationalities, Nanning 530006, China, 2016
14. Д. Місік, Ю. Зорін. Модифікований алгоритм квіткового запилення для розв’язання задачі глобальної оптимізації, Національний технічний університет України «Київський політехнічний інститут» ім. І. Сікорського, 2017.
15. Д. Місік, Ю. Зорін. Багатокритеріальний алгоритм оптимізації, Національний технічний університет України «Київський політехнічний інститут» ім. І. Сікорського, 2018.
16. C. Nagel, Professional C# 6 and .NET Core 1.0, O’Reilly Media, Wiley, New-York, 2016. – p. 131-135 – 1536 p.
17. J. Richter, Applied Microsoft Windows .NET Framework Programming, Microsoft Press, A Division of Microsoft Corporation, One Microsoft Way, Redmond, Washington, 2002
18. M. Mattew, “Pro ASP.NET 2.0 in C#”, Apress, 2005. – p. 21-23, 640 p.
19. E. Gamma, R. Helm, R. Johnson, J. Vlissidies, “Design Patterns: Elements of Reusable Object-Oriented Software”, Holland, 1994.
20. O. Wolf, “Introduction into Microservices”, Rolf the Warwick University, Bochum, Germany, 2014.
21. Virtual Library of Simulation Experiments: Test Functions and Datasets// Available at http://www.sfu.ca/~ssurjano/index.html.